ClubeDelphi

6 · Edição 77 · R$8,90

DevMedia GROUP

ISSN 1517990-7

Tudo sobre Validação de Dados

Mantenha dados consistentes no BD usando Triggers,Foreign Keys, Stored Procedures e UDF's

Técnicas de validação no ClientDataSet, DataSetProvider e Tfields

Validações em aplicações Web

Especial
oletos Bancários e artões de Crédito
mplemente a geração tomática de boletos no seu site e aprovação online de cartões de crédito

tando WebServices
emente testes automáticos ebservices Delphi

phi e UML
trua, na prática, um sistema do análise UML, POO e stência de Objetos – Parte II

Live Templates
Conheça esta nova ferramenta do Delphi para auxílio à escrita de código

Coluna IB/FB: Crie seu próprio SQLExplorer
Crie passo a passo uma ferramenta estilo "IBExpert" para operações de engenharia reversa de BD

Comprando esta edição tenha acesso na WEB a:

+180 vídeo aulas
+130 artigos

Assinatura

ClubeDelphi PLUS

Mais conteúdo DELPHI por muito menos!

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!
Confira o que você encontra no Portal do Assinante:
- Mais de 190 Vídeo Aulas!
- Diversos Mini-Cursos Gratuitos!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 138 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse www.devmedia.com.br/clubedelphi/portal.asp e utilize as informações abaixo:
Login: DVM.PL
Senha: FC73F

O acesso é válido por 30 dias a partida da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote ClubeDelphi PLUS, acesse: *http://www.devmedia.com.br/clubedelphiplus*

Corpo Editorial

Diretor Editorial e Administrativo
Gladstone Matos
gladstone@neoficio.com.br

Designer
Vinicius O. Andrade
viniciusoandrade@gmail.com

Ilustrador
Bruno Liserra
liserra3@hotmail.com

Capa
Antonio Xavier
jaxs_design@yahoo.com.br

Articulistas desta edição
Adriano Silveira, Adriano dos Santos, Fabrício Desbessel, Cristiano Caetano, Everson Borges Volaco e Paulo Quicoli

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Luciano Pimenta
lucianopimenta@clubedelphi.net

Revisão
Fabio Correa
fabiocorrea@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Publicidade
Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:
Kaline Dolabella
publicidade@devmedia.com.br

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Thiago Andrade – Atendimento ao Leitor
www.devmedia.com.br/central/default.asp
(21) 2220-5375

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2220-5375

Parcerias

Para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:
Jeff Wendell
jeff@devmedia.com.br

ASSESPRO PARANÁ

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão! Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

Luciano Pimenta - Editor do Site
lucianopimenta@clubedelphi.net

ClubeDelphi

Ano 6 - 77ª Edição - 2006 - ISSN 1517990-7 **Impresso no Brasil**

Editorial

Não importa o tipo de aplicação que você esteja desenvolvendo, seja ela Windows ou Web, precisará implementar formas eficazes para garantir a consistência das informações fornecidas pelos usuários do sistema. Existem inúmeras formas de validar dados em aplicações Delphi, e dependendo da arquitetura utilizada, vários locais onde a verificação pode ser implementada. Em uma solução multicamadas, por exemplo, podemos validar um dado no próprio banco, usando recursos como triggers e stored procedures, ou na camada servidora, usando métodos remotos ou eventos do DataSetProvider. A validação na aplicação cliente geralmente é mais limitada a verificações simples, como teste de campos nulos. Cada abordagem tem suas vantagens e desvantagens, depende de você desenvolvedor detectar a melhor forma e onde validar cada tipo de informação.

Nesta edição, Fabrício Desbessel e Gabriel Jank apresentam as várias formas de validar dados em aplicações Delphi. Saiba como garantir a consistência das informações no InterBase/Firebird usando campos Not Null, Check Constraints, Primary Keys, Unique Keys, Foreign Keys, Triggers, Stored Procedures, Exceptions e UDFs. Já na camada servidora, usamos o DataSetProvider para implementar consistências, no importante evento BeforeUpdateRecord, chamado como parte do ciclo de vida de atualização de um registro. O ClientDataSet recebe informações geradas por esse evento no OnReconcileError, onde temos várias opções de tratamento e recuperação de falhas. E finalmente, para aplicações Web, conheça os poderosos componentes de validação do ASP.NET, que permitem testar dados na própria interface cliente, nesse caso o browser, sem disparar requisições (postbacks) ao servidor, otimizado o tráfego de dados e melhorando a performance da aplicação. Tudo isso sem digitar uma linha de código!

Sucesso nos projetos Delphi e um abraço!

Guinther Pauli – *guinther@devmedia.com.br*
MCP, MCAD, MCSD.NET, Delphi Certified

ÍNDICE

06 - Aplicação Web completa - Parte IV
Luciano Pimenta
12 - Live Templates
Adriano Silveira
20 - Validações na prática
Fabrício Desbessel
32 - Testando Web Services
Cristiano Caetano
38 - Metadados no IB/FB
Everson Borges Volaco
46 - Mini-Curso de Delphi e UML – Parte II
Paulo Quicoli

FAÇA COMO A ALOG
FOQUE EM SEU NEGÓCIO

A ALOG é a única empresa brasileira totalmente focada em serviços de data center. Dedica-se 24 horas por dia, sete dias por semana.

ALOG desenvolveu no Brasil o conceito de **HOSTING GERENCIADO**: um serviço de excelência que garantirá a sua empresa estabilidade, agilidade, redundância, e escalabilidade. Para que isso seja verdade os clientes da ALOG contam com **Suporte Incondicional***, um conceito revolucionário em atendimento no Brasil.

Entre em contato com a nossa equipe.

NÍVEIS DE **HOSTING GERENCIADO**

PLANOS DE **HOSTING GERENCIADO**

GERENCIADO 1

- **Pentium D 2.8 GHz**
- 512 RAM DDR2
- 2 HD SATA 80 GB
- 450 GB de Transf. Mensal
- Firewall e Backup

BOOT REMOTO
GERENCIAMENTO**

R$**1.050,00** mensais

INSTALAÇÃO IMEDIATA***

GERENCIADO 2

- **Dual Xeon Dual Core 3.0 GHz**
- 1GB RAM DDR2 533
- 2 HD SATA RAID I 300 GB
- 1.000 GB de Transf. Mensal
- Firewall e Backup

BOOT REMOTO
GERENCIAMENTO**

R$**2.850,00** mensais

INSTALAÇÃO IMEDIATA***

* Sujeito a condições contratuais.
** Podendo escolher: HD, RD e GT
*** Sujeito a disponibilidade no estoque

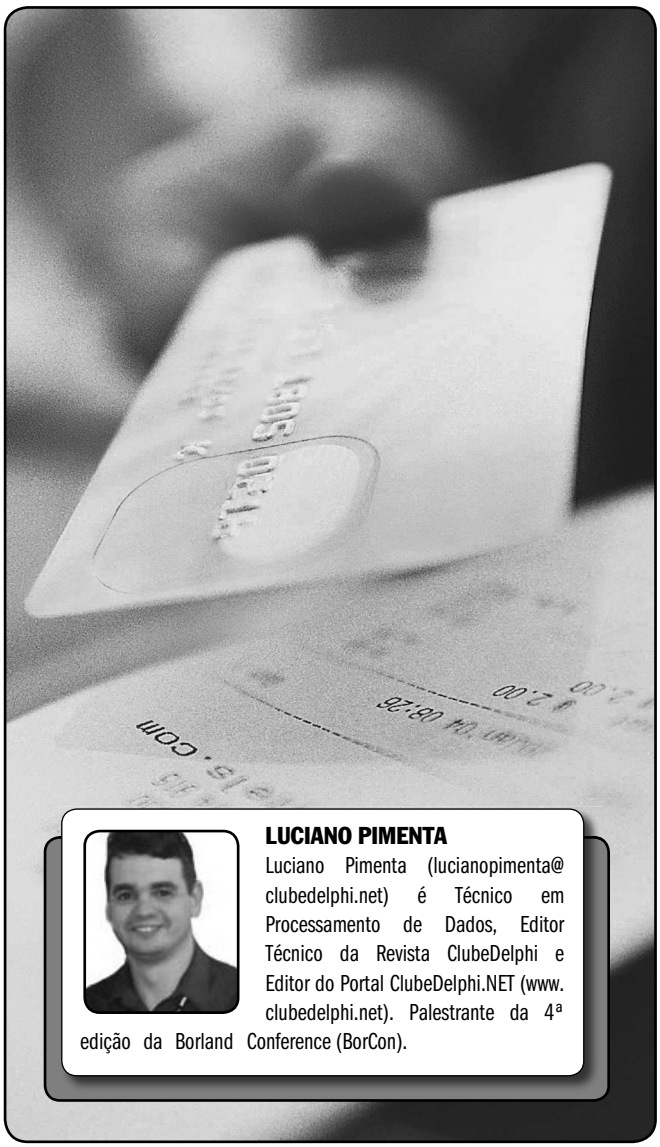
Hosting | Colocation | Contingência | Email Corporativo | Conectividade | Mainframes | AS400 | Outsourcing

ALOG data centers

Excelência em Hosting
www.alog.com.br :: 0800 282 3330

Aplicação Web Completa

Gerando boletos - Parte 4



LUCIANO PIMENTA

Luciano Pimenta (lucianopimenta@clubedelphi.net) é Técnico em Processamento de Dados, Editor Técnico da Revista ClubeDelphi e Editor do Portal ClubeDelphi.NET (www.clubedelphi.net). Palestrante da 4ª edição da Borland Conference (BorCon).

Neste artigo vamos concluir o nosso mini-curso de uma aplicação web completa, finalizando a compra do usuário no site, emitindo boleto bancário na aplicação ASP.NET. Daremos a opção de o mesmo ser enviado por e-mail. Para isso, usaremos uma ótima ferramenta que vai facilitar em muito a geração do boleto, o CobreBemX.

Imagine criar via código (no braço) um boleto bancário, com código de barras e para Web (ASP.NET), seriam necessárias umas 6 edições para descrevermos todo o processo! Com o CobreBemX, com algumas linhas de código, geramos boletos para vários bancos e também com logotipos personalizados da empresa. Além disso, podermos trabalhar com cartão de crédito (de várias bandeiras) e débito automático em conta corrente.

Abra a aplicação que construímos até agora e vamos começar a finalizar a aplicação web.

Baixando o CobreBemX

Existe uma versão para desenvolvimento e avaliação da ferramenta no site da empresa fabricante (veja seção *Links*). Com essa versão, existem duas restrições: a data de vencimento é fixa (com o ano de 2007), e o valor cobrado é mostrado como R$ 1,90.

Dica

Para rodar as páginas ASP.NET desse curso, é preciso que você tenha o Internet Information Services instalado. Para isso, você pode usar o Adicionar>Remover Programas do Painel de Controle. Caso tenha instalado o .NET Framework antes do IIS, é necessário executar a seguinte linha de comando:

C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322>aspnet_regiis.exe –i

Um sintoma comum da falta deste passo é que, no momento da execução da aplicação, os controles ASP.NET não aparecem no browser.

Baixe a versão de teste no endereço: *www.cobrebem.com/downloads/cbx/InstalarCobreBemX.zip* e descompacte o arquivo em uma pasta qualquer. Clique no arquivo *Instalar.exe* para registrar a DLL. O controle é um componente COM/DCOM e não será instalado no IDE do Delphi.

Existem mais algumas configurações e arquivos a serem baixados para gerar corretamente os boletos. Acesse o endereço *www.cobrebem.com/downloads/cbx/ImagensBoleto.zip* para baixar as imagens do boleto, essas imagens devem ser descompactadas na pasta da aplicação (crie uma pasta chamada "imagensboleto").

Não é recomendado alterar o nome dessas imagens, pois as mesmas serão usadas para o logotipo do banco, bem como o código de barras. Devemos no final gerar um arquivo, que será a licença de teste e o tipo de cobrança do boleto.

Acesse o endereço *www.cobrebem.com/cgi-bin/GeraArquivoLicencaTeste*, escolha o banco e em *Carteira*, escolha uma cobrança simples. Clique em *Gerar Arquivo* e salve o arquivo na pasta da aplicação (**Figura 1**).

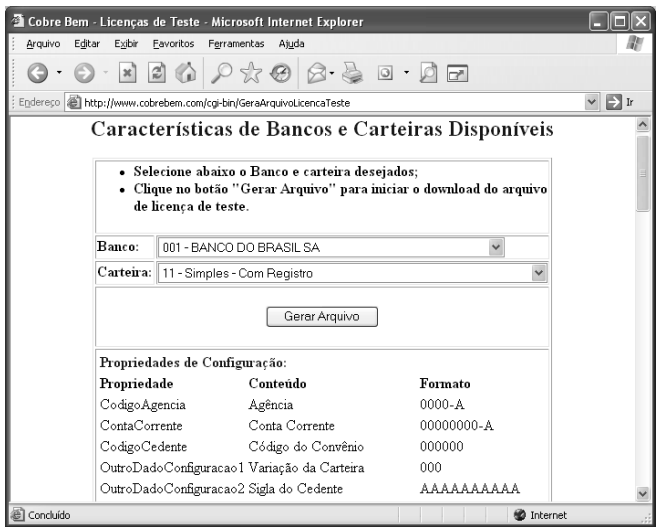


Figura 1. Gerando o arquivo de licença de teste e o tipo de carteira

> **Nota:** Para os testes deste artigo, a empresa fabricante gentilmente forneceu uma licença do componente, então os valores mostrados no boleto serão os valores reais da aplicação.

Usando o CobreBemX no Delphi

Caso não tenha aberto a aplicação, faça-a e acesse o arquivo *finaliza.aspx*. Adicione um *DropDownList* e através da propriedade *Items* adicione o seguinte valores: "Boleto", "Cartão de Crédito" e "Débito em conta".

A seguir, adicione um *CheckBox*, com o texto (propriedade *Text*): "Enviar boleto por e-mail" e um *Button* com o *Text* igual a "Confirmar Pagamento". Vamos habilitar o *CheckBox* apenas se a opção escolha no *DropDownList* for a de boleto. No evento *SelectedIndexChanged* do *DropDownList*, adicione o seguinte código:

```
CheckBox1.Enabled := DropDownList1.Items[0].Selected;
```

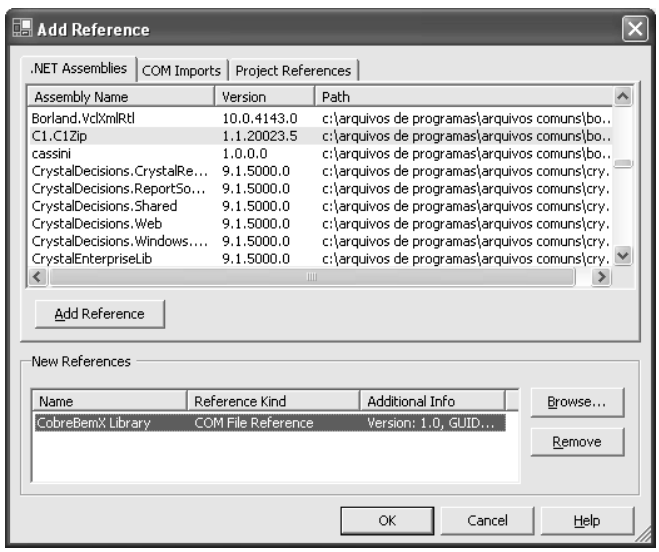


Figura 2. Adicionando a referência da DLL do CobreBemX

Altere a propriedade *AutoPostback* do *DropDownList* para *True*. Crie um novo *Web Form* dando o nome de "boleto.aspx". Clique com o botão direito no projeto, escolha a opção *Add Reference* e na janela aberta, clique em *Browse*. Escolha o arquivo *CobreBemX.dll* na pasta onde descompactou os arquivos do CobreBemX (**Figura 2**). Clique em OK. No *uses* do formulário adicione o *namespace CobreBemX*.

Gerando o boleto

A idéia aqui é receber o código do usuário, para que possamos fazer uma busca no banco para obter informações como endereço, nome completo etc., e também o código da venda, para que possamos saber o valor da venda.

Fica aqui uma dica: poderíamos passar o valor da venda através da URL, mas para usuários "mais espertos", basta alterar esse valor, assim o boleto será gerado com valores incorretos. Por isso, vamos ter o código da venda, para que façamos uma busca no banco, retornando os valores corretos.

Abra o "Data Module" que criamos anteriormente e adicione dois *FbCommands* ("fbUsuario" e "fbVenda"). Faça a ligação com o *FbConnection* e para o código SQL, selecione todos os dados das tabelas USUARIOS e VENDAS, parametrizando pelo campo chave, semelhante ao que fizemos em outros exemplos do curso.

Não esqueça de adicionar os parâmetros nos *FbCommands*. No código vamos adicionar dois métodos, conforme a **Listagem 1.**

Listagem 1. Criando os métodos para retornar os dados do usuário e venda

```
public
...
  function GetUsuario (aIdUsuario: string): FbDataReader;
  function GetVenda (aIdVenda: string): FbDataReader;
...
function TDM.GetUsuario(
  aIdUsuario: string): FbDataReader;
var
  dr: FbDataReader;
begin
  FbConnection1.Open;
  fbUsuario.Parameters[0].Value := aIdUsuario;
  dr := fbUsuario.ExecuteReader(CommandBehavior.CloseConnection);
  Result := dr;
end;

function TDM.GetVenda(aIdVenda: string): FbDataReader;
var
  dr: FbDataReader;
begin
  if FbConnection1.State = ConnectionState.Closed then
    FbConnection1.Open;
  fbVenda.Parameters[0].Value := aIdVenda;
  dr := fbVenda.ExecuteReader(CommandBehavior.CloseConnection);
  Result := dr;
end;
```

O código é bem simples, onde passamos como parâmetro os códigos do usuário e vendas e recebemos um *fbDataReader*. Note que no *GetVendas* verificamos se a conexão já está fechada, para chamarmos o *Open* do *FbConnection*.

Isso é interessante, pois se existir uma conexão aberta (do usuário), vai ser gerada uma mensagem de erro. Com isso, vamos conectar somente se não tivermos conexões pendentes.

Voltando à página *boleto.aspx*, vamos implementar o método "GeraBoleto", que terá o código da **Listagem 2**.

Entendendo o código da geração do boleto

Vamos entender o código, onde primeiramente, declaramos as variáveis necessárias, após, carregamos os dados do usuário e da venda. Iniciamos a configuração do boleto passando para a propriedade *ArquivoLicenca* o caminho do arquivo gerado no site.

A seguir, temos a configuração dos dados da conta corrente da empresa (no nosso exemplo a DevMedia Shop), ou seja, esses dados são para saber para quem o usuário do site está pagando. Depois dessas informações, configuramos um logotipo para o boleto (para que a imagem não fique pequena ou grande, o tamanho ideal é: 100x100) e também o caminho onde estão as imagens que baixamos no site, necessárias para o logotipo do banco e o código de barras.

A seguir, temos as informações do cliente que comprou no site, dados esses oriundos da consulta ao banco. Alguns foram adicionados diretamente no código (caso do CPF), mas em uma aplicação real, esses dados viriam do banco, claro.

A data do documento é a do dia, e a data de vencimento para daqui a 10 dias. Essas informações podem também ser facilmente resgatadas do banco de dados, dando mais flexibilidade na geração do boleto, podendo ser configurada pelo administrador do site.

Após, preenchemos o número e valor do documento, e podemos adicionar informações, como: dados de cobrança de juros e outras necessárias, através das propriedades *Demonstrativo* e *InstrucoesCaixa*.

Finalmente, para gerar o boleto, usamos o *Response.Write*, passando como parâmetro o código do controle. No final, liberamos a variável da memória e fechamos a conexão com o banco, através do *FbDataReader*. Volte a página do carrinho e altere a URL para abrir o *finaliza.aspx*, onde adicionaremos o código da venda:

Listagem 2. Método para gerar o boleto

```
procedure TWebForm1.GeraBoleto;
var
  aDM: TDM;
  adrUsuario, adrVenda: FbDataReader;
  aCobreBem: CobreBemX.ContaCorrenteClass;
  aBoleto: CobreBemX.IDocumentoCobranca;
begin
  aDM := TDM.Create;
  adrUsuario := aDM.GetUsuario(
    Session['ID_USUARIO'].ToString);
  adrVenda := aDM.GetVenda(
    Request.QueryString['Venda'].ToString);
  adrUsuario.Read;
  adrVenda.Read;

  aCobreBem := CobreBemX.ContaCorrenteClass.Create;

  with aCobreBem do
  begin
    { arquivo de licença gerado no site }
    ArquivoLicenca := Server.MapPath('001-11.conf');

    { dados da conta corrente - DevMedia Shop }
    CodigoAgencia := '1234-5';
    NumeroContaCorrente := '000001234-X';
    CodigoCedente := '123456';
    OutroDadoConfiguracao1 := '019';
    InicioNossoNumero := '00001';
    FimNossoNumero := '99999';
    ProximoNossoNumero := '00015';

    PadroesBoleto.PadroesBoletoEmail.URLLogotipo :=
      'http://localhost/CursoASPNET/images/'+
      'logoboleto.gif';
    PadroesBoleto.PadroesBoletoEmail.
      URLImagensCodigoBarras := 'http://localhost/'+
        'CursoASPNET/imagensboleto/';

    aBoleto := DocumentosCobranca.Add;
  end;

  with aBoleto do
  begin
    { Monta os dados do sacado, nosso cliente }
    NomeSacado := adrUsuario['NOME_USUARIO'].ToString;
    CPFSacado := '111.111.111-11';
    BairroSacado := 'Bairro';
    EnderecoSacado := adrUsuario['ENDERECO'].ToString;

    {Data do documento, hoje }
    DataDocumento := DateTime.Now.ToShortDateString;

    { Data de Vencimento, daqui a 10 dias }
    DataVencimento :=
      DateTime.Now.AddDays(10).ToShortDateString;

    NumeroDocumento := '123';
    ValorDocumento :=
     Convert.ToDouble(adrVenda['TOTAL_VENDA']);

    PadroesBoleto.Demonstrativo :=
      'Compra na Loja DevMedia Shop';
    PadroesBoleto.InstrucoesCaixa :=
      'Cobrar juros de 1% após 5 dias de atraso';
  end;

  { gera o boleto }
  Response.Write(aCobreBem.get_GeraHTMLBoleto(0));

  { libera a variável da memória }
  aCobreBem := nil;

  { fecha a conexão com o banco,
    através do FbDataReader }
  adrUsuario.Close;
  adrVenda.Close;
end;
```

```
...
Session['CARRINHO'] := nil;
Response.Redirect('finaliza.aspx?Venda='+ IdCompra.ToString);
```

Como o código do usuário está em sessão, não precisamos utilizar o mesmo na URL. Na página *finaliza.aspx*, adicione o seguinte código no botão para confirmar o pagamento:

```
if DropDownList1.Items[0].Selected then
  Response.Redirect('boleto.aspx?Venda='+
    Request.QueryString['Venda'].ToString);
```

Por fim, no *Load* da página *boleto.aspx*, adicione o seguinte código:

```
if Request.QueryString['Venda'] <> nil then
  GeraBoleto;
```

Rode a aplicação, realize as compras e teste a geração do boleto. Nas **Figuras 3** e **4** temos o boleto gerado. Nas figuras estão marcados os dados oriundos da aplicação, como: data de vencimento, valor, nome e endereço do usuário.

Enviando o boleto por e-mail

Vimos que ao configurar corretamente as propriedades geramos um boleto pronto para ser usado. E se o usuário do site quiser receber o mesmo pelo e-mail? A facilidade de envio de e-mail é a mesma.

No formulário do boleto, vamos criar mais um método, responsável pelo envio do e-mail, conforme o código a seguir:

```
procedure EnviaEmail (aCobreBem: CobreBemX.
  ContaCorrenteClass;
  aBoleto: CobreBemX.IDocumentoCobranca;
  aEmail, aNome: string);
```

Para não duplicar o código do boleto, vamos usar na *procedure*, como parâmetro, as váriaveis da geração do boleto. Passamos também por parâmetro o e-mail e o nome do usuário. Na **Listagem 3** temos a implementação do código para envio de e-mail.

Listagem 3. Enviando o boleto por e-mail

```
procedure TWebForm1.EnviaEmail(aCobreBem: CobreBemX.ContaCorrenteClass;
  aBoleto: CobreBemX.IDocumentoCobranca;
  aEmail, aNome: string);
begin
  { configurações do servidor de e-mail }
  with aCobreBem.PadroesBoleto.PadroesBoletoEmail do
  begin
    SMTP.Servidor := 'servidor de e-mail';
    SMTP.Porta := 25;
    SMTP.Usuario := 'nome do usuario';
    SMTP.Senha := 'senha';
    URLImagensCodigoBarras := 'http://localhost/'+
      'CursoASPNET/imagensboleto/';
    URLLogotipo := 'http://localhost/CursoASPNET/images/'+
      'logoboleto.gif';
    PadroesEmail.Assunto := 'Boleto de cobrança';
    PadroesEmail.EmailFrom.Endereco := aEmail;
    PadroesEmail.EmailFrom.Nome := aNome;
    PadroesEmail.FormaEnvio := FormaEnvioEmailEnum.feeSMTPBoletoHTML;
  end;

  with aBoleto do
  begin
    EnderecosEmailSacado.Add.Nome := aNome;
    EnderecosEmailSacado.Add.Endereco := aEmail;
    ControleProcessamentoDocumento.EnviaEmail :=
      StatusControleProcessamentoEnum.scpExecutar;
  end;

  { envia o e-mail }
  aCobreBem.EnviaBoletosPorEmail;
end;
```

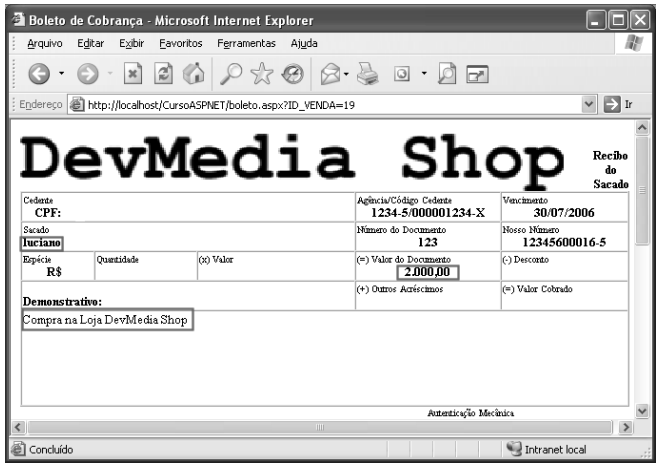


Figura 3. Boleto gerado na aplicação ASP.NET (parte superior)

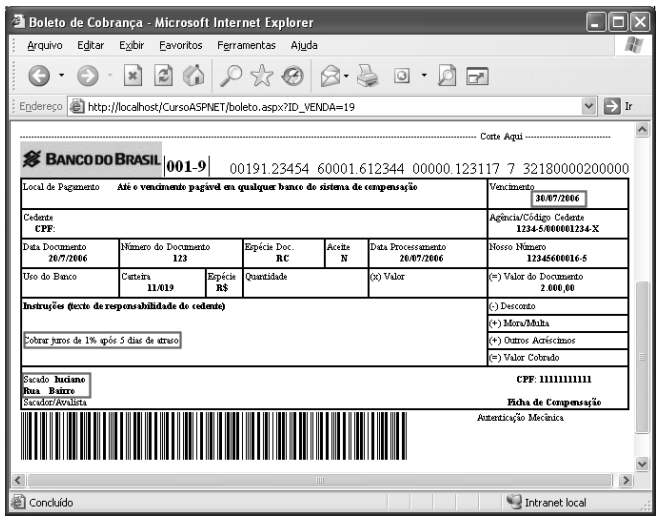


Figura 4. Parte inferior do boleto gerado na aplicação

Vamos alterar mais uma vez a chamada à página do boleto (na página *finaliza.aspx*), para que inclua a informação se o usuário deseja enviar o boleto por e-mail. Altere a chamada para o seguinte código:

```
if DropDownList1.Items[0].Selected then
  Response.Redirect('boleto.aspx?Venda='+
    Request.QueryString['Venda'].ToString+'&Send='+
    CheckBox1.Checked.ToString);
```

Estamos incluindo na URL mais um parâmetro, indicando se o boleto deve ser enviado por e-mail. Na página do boleto, no método *GeraBoleto* altere o código, após a geração do boleto, conforme a **Listagem 4**.

Listagem 4. Alterando o método GeraBoleto

```
...
{ gera o boleto }
Response.Write(aCobreBem.get_GeraHTMLBoleto(0));

if Request.QueryString['Send'] <> nil then
  EnviaEmail(aCobreBem, aBoleto,
    adrUsuario['EMAIL'].ToString,
    adrUsuario['NOME_USUARIO'].ToString);
...
```

Rode a aplicação e teste o envio do e-mail (**Figura 5**).

Pagamento com Cartão de Crédito

Podemos realizar o pagamento das compras na aplicação Web através de cartão de crédito. No Delphi, vamos criar as páginas para aprovação ou não do resultado da consulta do cartão. Crie dois *Web Forms*, dando o nome de: "aprovado.aspx" e "reprovado.aspx". Nessa última página, apenas adicione o seguinte código no *Load*:

```
Response.Write('Infelizmente seu cartão <b> não '+
  '</b> foi aprovado');
Response.Write('<br><b>Motivo:</b>');
Response.Write('<br>'+Request.QueryString[
  'Motivo'].ToString);
```

Na página *finaliza.aspx*, adicione um *Panel* e altere código no botão de confirmação de pagamento:

```
...
Panel1.Visible := DropDownList1.Items[1].Selected
```

Na página de aprovação do cartão, adicione o código da **Listagem 5** no evento *Load* da página:

```
Response.Write('Pagamento <b> Aprovado!</b>'+
  '<br>Obrigado por comprar na DevMedia Shop!');
Response.Write('<br><br><b>Comprovante:</b>');
Response.Write('<br>'+Request.QueryString['Comprovante'].ToString);
Response.Write('<br><br><b>Resultado:</b>');
Response.Write('<br>'+Request.QueryString['Resultado'].ToString);
```

Vamos agora criar o layout para que o usuário adicione os dados do cartão de crédito. Algum desenvolvedor mais esperto pode dizer: "Posso ter esses dados do cartão no banco, onde salvo na primeira compra do cliente e depois apenas busco os mesmos através de uma consulta simples. Assim o usuário não precisa adicionar tudo de novo!".

Por uma questão de comodidade do usuário, até que a idéia é boa, mas lembre-se que os dados do cartão de crédito do cliente são informações muito importantes e de extrema responsabilidade. Pois, caso aconteça algum problema, e o mesmo seja acessado por pessoas desautorizadas, você terá muita dor de cabeça.

Altere para *False* a propriedade *Visible* do *Panel* e adicione alguns componentes visuais (dentro do *Panel*). Tome por base a **Figura 6** (veja na propriedade *Text* o nome dos *TextBoxs*).

No botão *Solicita* adicione o código da **Listagem 6.**

Listagem 6. Código para aprovação do cartão de crédito

```
var
  aCobreBem: CobreBemX.ContaCorrenteClass;
  aCartao: CobreBemX.IDocumentoCobranca;
  aDM: TDM;
  dr: FbDataReader;
begin
  aDM := TDM.Create;
  dr := aDM.GetVenda(Request.QueryString[
    'Venda'].ToString);
  dr.Read;
  aCobreBem := CobreBemX.ContaCorrenteClass.Create;

  { arquivo de licença gerado no site }
  with aCobreBem do
  begin
    ArquivoLicencaCartoesCredito :=
      Server.MapPath('001-11.conf');

    { moacTeste = Teste,
      moacSimulacao = Simulação,
      moacProducao = Produção }
    PadroesAprovacaoCartoes.ModoOperacao :=
      ModoOperacaoAprovacaoCartaoEnum.moacTeste;
    DocumentosCobranca.Clear;
  end;

  { dados do cartão de crédito }
  aCartao := aCobreBem.DocumentosCobranca.Add;

  with aCartao do
  begin
    CartaoCredito.MesValidade :=
      Convert.ToInt16(txtMes.Text);
    CartaoCredito.AnoValidade :=
      Convert.ToInt16(txtAno.Text);
    ValorDocumento := 1.99;
    CartaoCredito.QuantidadeParcelas :=
      Convert.ToInt16(txtParcelas.Text);
    CartaoCredito.Numero := txtCartao.Text;
    CartaoCredito.CodigoSeguranca := txtCode.Text;
  end;

  { Solicita a aprovação do cartão }
  if aCobreBem.SolicitaAprovacaoCartao[0] then
  begin
    { solicitação aprovada }
    aCobreBem.ConfirmaAprovacaoCartao[0];
    Response.Redirect('aprovado.aspx?Comprovante='+
      aCartao.CartaoCredito.ComprovanteAdministradora+
      '&Resultado='+aCartao.CartaoCredito.
      ResultadoSolicitacaoAprovacao)
  end
  else
    { cartão reprovado }
    Response.Redirect('reprovado.aspx?Motivo='+
      aCartao.CartaoCredito.
      ResultadoSolicitacaoAprovacao);

  aCobreBem.DocumentosCobranca.Clear;
  aCobreBem := nil;
end;
```

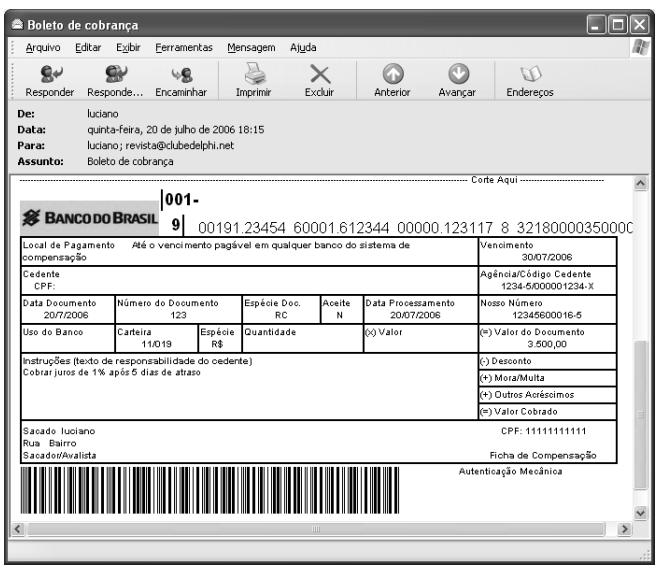


Figura 5. Boleto gerado na aplicação e enviado por e-mail

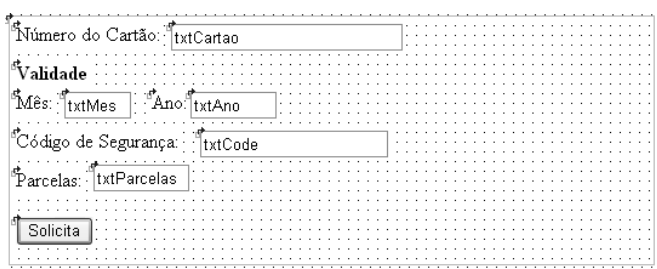


Figura 6. Panel para adicionar os dados do cartão

O código é bastante semelhante ao da geração do boleto, porém estamos usando a interface *ICartaoCredito*. Após ter preenchido as propriedades, chamamos o método *SolicitaAprovacaoCartao*, que retorna um *Boolean*, onde verificamos o valor e redirecionamos para a página de aprovação, com o valor do comprovante da administradora, ou reprovação, com o motivo.

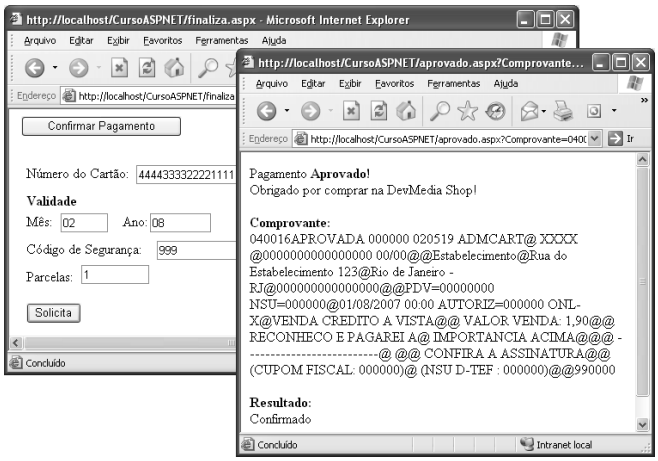


Figura 7. Aprovação da compra

Na **Figura 7** temos os dados de um cartão de teste e o resultado de aprovação.

Conclusão

Terminamos nesta edição o mini-curso de criação de uma aplicação Web completa, usando Delphi 2006 e ASP.NET. Mostramos várias dicas, para que o desenvolvedor que está iniciando em ASP.NET ou já possui uma base teórica, possa criar aplicações Web robustas e com facilidade. Neste artigo, mostramos como é fácil, através de uma ferramenta, gerar boletos e cobranças com cartão de crédito, dando comodidade para o usuário. Um grande abraço a todos e até a próxima! ■

Links

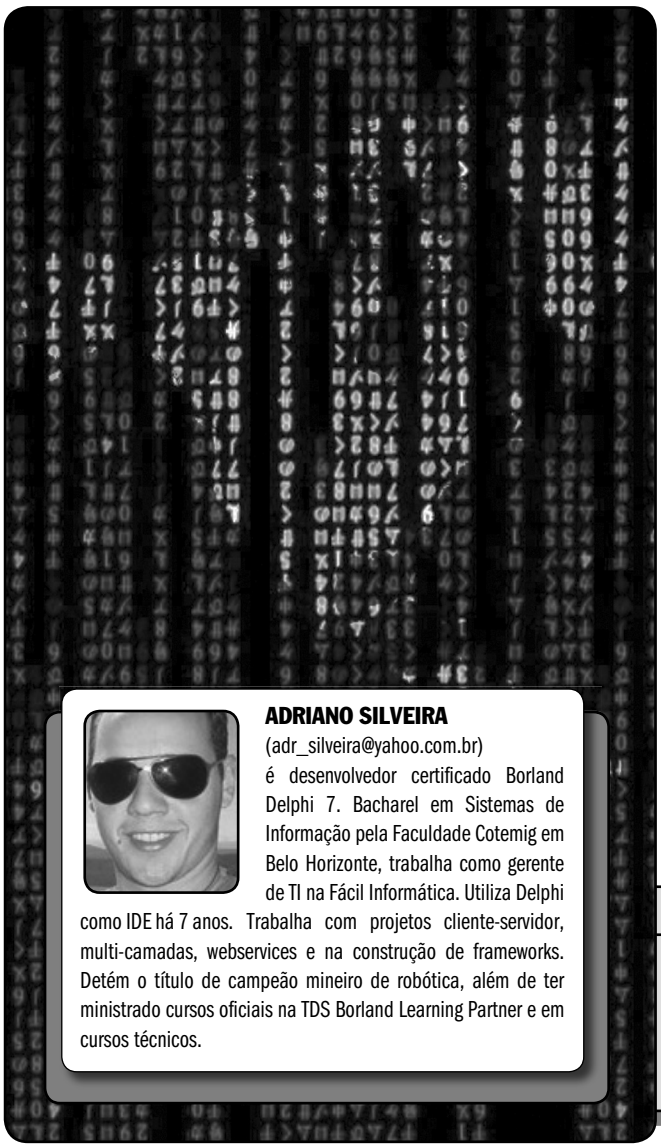
Curso de E-Commerce com Delphi 2005 e ASP.NET

www.devmedia.com.br/curso/ecommerce2005

CobreBem

www.cobrebem.com.br

Live Templates



ADRIANO SILVEIRA
(adr_silveira@yahoo.com.br)
é desenvolvedor certificado Borland Delphi 7. Bacharel em Sistemas de Informação pela Faculdade Cotemig em Belo Horizonte, trabalha como gerente de TI na Fácil Informática. Utiliza Delphi como IDE há 7 anos. Trabalha com projetos cliente-servidor, multi-camadas, webservices e na construção de frameworks. Detém o título de campeão mineiro de robótica, além de ter ministrado cursos oficiais na TDS Borland Learning Partner e em cursos técnicos.

Desde a versão 3 do Delphi, podemos utilizar o recurso de *Code Templates*, que forneciam blocos de código comumente utilizados em nossos projetos. Contudo, eram bastante limitados se resumindo a fornecer uma cópia de um bloco de código "colado" no editor.

Além disso, o único recurso extra existente era a possibilidade de indicar o ponto onde o cursor seria posicionado no momento da utilização do *template* de código, através do character "|" (*pipe*). No Delphi 2006, o conceito de *Code Templates* foi alterado para *Live Templates*, isso porque diversos recursos foram adicionados, como veremos neste artigo.

Utilizando Live Templates

Para utilizar um *Live Template* pressione CTRL+J no ponto do código onde deseja inseri-lo. A lista com todos os *Live Templates* será exibida. Selecione o *template* desejado e pressione ENTER. Outra maneira de utilizar um *Live Template* é selecionar o menu *View>Templates* (**Figura 1**).

Para adicionar um *Live Template* através da janela, dê um duplo clique sobre o *template* desejado. Ao adicionar um *template* ao código já podemos perceber o primeiro novo recurso: é possível se movimentar entre pontos de código utilizando a tecla TAB.

Por exemplo, adicione o *template Class* e pressionando TAB, e veja que é possível alternar entre a declaração do nome da classe e sua ancestral.

CLUBEDELPHI PLUS!

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como criar um Live Template para acesso pronto a um banco de dados Firebird.

http://www.devmedia.com.br/articles/visualizacomponente2.asp?comp=2510

Grupos de Templates

Devido ao fato do Delphi 2006 possuir múltiplas personalidades, os *Live Templates* foram agrupados de acordo com a linguagem (Delphi, C#, C++, etc.). Além desses, existe também o grupo IDE, fornecendo um único *Template*, utilizado na construção de novos.

Outra novidade interessante é o recurso de automação de escrita de código. Por exemplo, insira o *template "for"* em algum ponto do código. Repare que o ponto onde o cursor se inicia é exatamente o nome da variável que será utilizada na iteração.

Altere seu nome para "Aux" e pressione ENTER. Observe que o próprio IDE declara a variável *Aux*. Esse recurso é válido para outros *Live Templates* como *for in* e *try/finally*.

Conhecendo os Live Templates

Nesse ponto serão apresentados e descritos os *Live Templates* disponíveis para o Delphi. A maioria desses possui correspondentes para as linguagens C# e C++, como por exemplo: *case* do Delphi com *switch* do C# e do C++. Veja na **Tabela 1** a descrição dos *Lives Templates* presentes no Delphi 2006.

Código dos Live Templates

Em algumas situações, possivelmente, nenhum dos *Live Templates* seja compatível com alguma necessidade de código, assim é possível criar nossos próprios *templates*. Todos são escritos no padrão XML, como podemos ver o exemplo da **Listagem 1**.

Listagem 1. Exemplo de Live Templates

```
<?xml version="1.0" encoding="utf-8" ?>
<codetemplate
  xmlns="http://schemas.borland.com/Delphi/2005/codetemplates"
  version="1.0.0">
    <template name="tryf" surround="true" invoke="manual">
        <description>
            try finally
        </description>
        <author>
            Borland Software Corporation
        </author>
        <code language="Delphi" context="methodbody"
                      delimiter="|"><![CDATA[try
                      |selected||*||end|
                      finally
                      end;
                      ]]>
        </code>
    </template>
</codetemplate>
```

Como não é o objetivo do artigo, não entraremos nos detalhes de padrões XML, concentrando-se apenas no que diz respeito à criação de *Live Templates*, pré-supondo que seja de conhecimento do leitor as regras e padrões XML.

Conforme visto na listagem anterior, a primeira linha contém o cabeçalho do arquivo. Após, a primeira *tag* é a *codetemplate*, que delimita o início e o término do *Live Template* e contém os seguintes atributos:

- *xmlns*: contém o *namespace* do XML;
- *version*: versão do *Live Template*.

Todo *Live Template* contém essa *tag* com os respectivos atributos. A primeira *tag* interna do *Live Template* é a <*template*> que define os seguintes atributos globais:

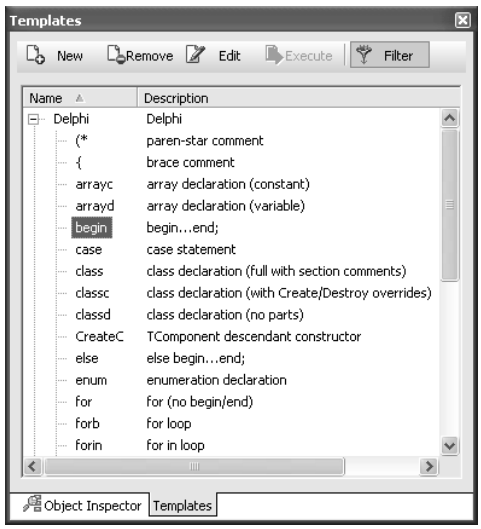


Figura 1. Janela Templates do Delphi 2006

- *name*: nome do *Live Template*;
- *surround*: define se será disponibilizado na lista de *Sorrounds* (veja box);
- *invoke*: define a forma como o *Live Template* será "invocado", que pode ser:
    - *auto*: pressionando SPACE ou TAB;
    - *manual*: pressionando TAB;
    - *none*:pressionandoCTRL+J,CTRL+SPACE ou usando a janela *Template Viewer*.

A *tag description* é opcional e define uma descrição do *Live Template*. A *author* também é opcional e logicamente, define o criador do *Live Template*. A *tag* que realmente contém o código do *Live Template* é a *code*, com os seguintes atributos:

- *language*: obrigatório. Define a linguagem que o *Live Template* pertence, podendo ser: Delphi, CSharp, C, XML, HTML, Jsharp, IDE, SQL, IDL, JavaScript, CSS, INI, PHP.
- *context*: contexto de visibilidade do código, que pode ser:
    - *memberdecl*: declaração de membro, inserido entre a declaração da classe e seu término;
    - *methoddecl*: declaração de método;
    - *decl*: declaração de variável;
    - *typedecl*: declaração de tipo (*type*);
    - *methodbody*: corpo de método;
    - *page*: reservado para uso futuro;
    - *file*: reservado para uso futuro;
    - *any*: inserido em qualquer contexto;

Sorround

Recurso incorporado na versão 2006, que insere um bloco de código selecionado para "dentro" de uma estrutura de código selecionada, como while, for e begin/end.

- *delimiter*: define o character delimitador de pontos de tabulação;
- *![CDATA]*: define o bloco de código "colado" no editor.

Para criar pontos de tabulação é preciso defini-los antes da *tag code*. Para criar um ponto de tabulação é preciso adicionar uma *tag point* com os seguintes atributos/sub-tags:

- *name*: nome do ponto de tabulação;
- *default*: o ponto que possuir esse valor como verdadeiro será o primeiro ponto selecionado;
- *editable*: define se o ponto poderá ser acessado durante a navegação;
- *text*: *tag* opcional que define um valor padrão para o ponto de tabulação;
- *hint*: texto que é exibido como *hint* no momento da entrada no ponto de tabulação.

Conjunto de Characters

Evite a utilização de characters acentuados. O interpretador não é *unicode* e erros podem ocorrer se usarmos characters acentuados.

Além de montar os pontos de tabulação é possível utilizar alguns scripts para validar o *Live Template*, declarar variáveis e invocar o *Code Completion*. Para utilizar um script é preciso adicionar uma *tag script* dentro ou fora de uma *point*.

Além disso, é preciso definir o momento da chamada de sua execução. Por exemplo, para executar a declaração de uma variável no momento da saída do *Live Template* (pressionando a tecla TAB), o seguinte código deve ser adicionado:

Live Template	Permite navegação	Declara variável	Descrição
(*	Não	Não	Insere um conjunto de characters para criação de bloco de comentários.
{	Não	Não	Mesma funcionalidade do anterior.
ArrayC	Sim	Não	Cria um array de "n" posições com valores constantes.
ArrayD	Sim	Não	Cria um array de "n" posições com valores variáveis.
Begin	Não	Não	Insere um bloco begin/end no código.
Case	Sim	Não	Insere uma instrução case simples.
Class	Sim	Não	Insere o corpo completo de declaração de classe, incluindo comentários.
ClassC	Sim	Não	Igual ao anterior somado à métodos construtores e destrutores.
ClassD	Sim	Não	Insere o corpo simples de declaração de classe.
CreateC	Não	Não	Insere declaração de um método construtor.
Else	Não	Não	Insere bloco de código com instrução else/begin/end.
Enum	Sim	Não	Cria o corpo de construção de um tipo enumerado.
For	Sim	Sim	Insere corpo de código da instrução for.
ForB	Sim	Sim	Igual ao anterior somado ao bloco de código begin/end.
ForIn	Sim	Sim	Insere corpo de código da instrução for in.
ForR	Sim	Sim	Insere corpo de código da instrução for reverso (downto).
Function	Sim	Não	Insere corpo de código para criação de funções.
If	Sim	Não	Insere corpo de código simples da instrução if.
IfB	Sim	Não	Igual ao anterior somado ao bloco de código begin/end.
IfE	Sim	Não	Igual ao anterior somado ao bloco de código else simples.
IfEB	Sim	Não	Igual ao anterior somado ao bloco de código else/begin/end.
Procedure	Sim	Não	Insere corpo de código para criação de procedures.
Region	Sim	Não	Cria uma region nomeada.
Repeat	Sim	Não	Insere corpo de código simples da instrução repeat/until.
Try	Sim	Sim	Insere corpo de código para criação dinâmica de objetos. Além de inserir código de criação, utiliza try/finally para destruição garantida do objeto.
TryE	Sim	Não	Insere corpo de código para tratamento de exceções incluindo declaração do objeto de exceção.
TryF	Sim	Não	Insere corpo de código simples da instrução try/finally.
Var	Sim	Sim	Insere código para declaração simples de variável.
While	Sim	Não	Insere corpo de código simples da instrução while.
WhileB	Sim	Não	Igual ao anterior somado ao bloco de código begin/end.

Tabela 1. Descrição dos Live Templates disponíveis para o Delphi 2006

```
<script language="Delphi" onenter="false" onleave="true" onvalidate="false">
DeclareVariable(|var|);
</script>
```

Neste exemplo temos a declaração da variável *var* que na verdade é um ponto de tabulação no momento da "saída" do *Live Template* (*onleave*). Existem alguns outros eventos, conforme podemos observar na **Tabela 2**.

Evento	Descrição
onvalidate	Ocorre antes do script ser inserido no código.
onenter	Ocorre quando um ponto de código recebe foco.
onleave	Ocorre quando um ponto de código perde foco.

Tabela 2. Eventos de script

Além de declarar variáveis, é possível executar outras ações dentro dos scripts (**Tabela 3**).

Função	Linguagem
InvokeClassCompletion	Delphi
PopulateCase(|expression|)	Delphi
DeclareVariable(|variable|)	Delphi
RemoveTemplate	Delphi
ValidateForTemplate	Delphi
GetClipboardContents	Delphi
VarDeclToAssignment	Delphi
Invoke_code_completion	C
Populate_case($expr$)	C

Tabela 3. Funções de script

Criando Live Templates

Para criar um *Live Template*, acesse dentro do *Object Repository*, o grupo *Other Files* e em seguida o item *Code Template* (**Figura 2**).

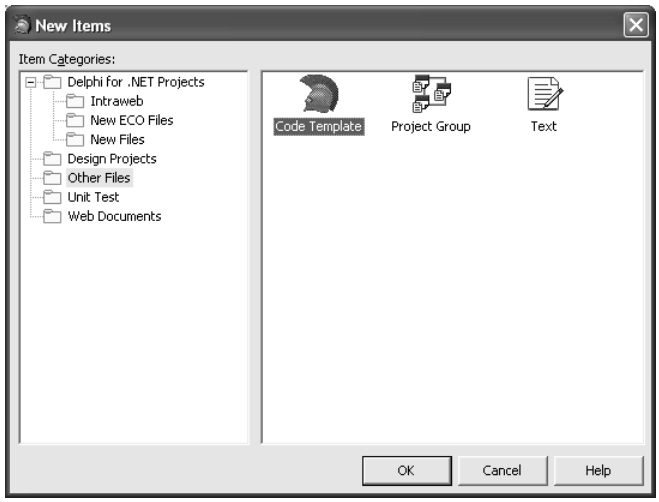


Figura 2. Criando um novo Template a partir do Object Repository

Identação

Para explicitar a identação de uma linha de código dentro da sessão *![CDATA[]]*, utilize |*| "n" vezes antes de cada linha. Cada |*| produzirá um nível de identação.

Navegação entre pontos

Para que o interpretador do BDS consiga efetuar a navegação entre os pontos de código (*point*), é necessário definir o *character* delimitador. Por padrão o character "$" é utilizado. Contudo, esse pode gerar conflitos com outros códigos, portanto, deve-se definir outro. Normalmente utiliza-se o character pipe ("|").

Ao confirmar a operação, uma nova janela de código será criada com o código inicial do *Live Template*, como podemos ver na **Listagem 2**.

Aceite **pagamentos on-line** em seu **Website** em minutos

Sem convênio. Sem taxa de adesão. Sem taxa mensal. Sem período mínimo.

BRpay.com.br Pagamentos

VISA AMEX Itaú Shopline Boleto Depósito Bancário *Aceite pagamentos online em minutos*

Amplie seus horizontes Alcance novos compradores

A BRpay oferece um rápido, acessível e conveniente serviço de pagamentos online, para Web sites e comerciantes de qualquer porte e tamanho com um baixo custo. A BRpay não cobra taxas para abertura de conta, não cobra taxa de manutenção, não requer prazo mínimo de utilização ou volume mínimo. Você só paga uma pequena taxa sobre o que recebe, ou seja, se não recebe nada, então você não paga nada.

www.BRpay.com.br

Listagem 2. Código inicial do Live Template

```
<?xml version="1.0" encoding="utf-8" ?>
<codetemplate
   xmlns="http://schemas.borland.com/Delphi/2005/codetemplates"
     version="1.0.0">
       <template name="" invoke="manual">
          <description>
          </description>
          <author>
          </author>
          <code language=""><![CDATA[]]>
          </code>
</template>
</codetemplate>
```

Definições Iniciais

O primeiro passo é definir o nome do *Live Template*. Atribua o valor "populalistview" para o atributo *name* da *tag template*. Em seguida atribua o valor "Delphi" para *language* em *code*. Esse atributo definirá o grupo de linguagens onde o *Live Template* participará e será exibido.

Por fim, crie um novo atributo "delimiter" com o valor "|" dentro da tag *code*. O mesmo definirá o character delimitador de "pontos de código". Por fim, preencha o conteúdo das *tags description* e *author* com os valores "Template para popular um ListView a partir de um DataSet" e "ClubeDelphi", respectivamente. O resultado está na **Listagem 3.**

Listagem 3. Código do Live Template após as definições iniciais

```
<?xml version="1.0" encoding="utf-8" ?>
<codetemplate
  xmlns="http://schemas.borland.com/Delphi/2005/codetemplates"
   version="1.0.0">
    <template name="populalistview" invoke="manual">
          <description>
              Template para popular um ListView a partir de um DataSet
          </description>
          <author>
              ClubeDelphi
          </author>
          <code language="Delphi" delimiter="|"><![CDATA[]]>
          </code>
</template>
</codetemplate>
```

Armazenamento

O BDS reconhece todos os arquivos XML do diretório *Application Data\Borland\BDS\4.0\Code_Templates* como *Live Templates*, desde que sigam as regras de sintaxe. Além desse, o diretório *Borland\BDS\4.0\Objrepos\Code_Templates\delphi* da instalação do BDS também é reconhecido.

Nesse último estão os *Templates* do BDS, então salve o nosso *template* nesse diretório, com o nome de "populalistview.xml".

Pontos de Código

Para utilizar o novo *template* é necessário adicionar os pontos de código contidos na **Tabela 4**.

Listagem 4. Código do Live Templates, após a adição dos pontos de código

```
<?xml version="1.0" encoding="utf-8" ?>
<codetemplate xmlns="http://schemas.borland.com/
  Delphi/2005/codetemplates" version="1.0.0">
    <template name="populalistview" invoke="manual">
        <description>
            Template para popular um ListView a partir de um DataSet
        </description>
        <author>
            Clube Delphi
        </author>
        <point name="NewItem">
            <text> NewItem </text>
            <hint>
                Nome do objeto "TListItem" que contera as instancias
                dos itens do "ListView"
            </hint>
        </point>
        <point name="NewColumn">
            <text> NewColumn </text>
            <hint>
              Nome do objeto "TListColumn" que contera as colunas do "ListView"
            </hint>
        </point>
        <point name="DataSet">
            <text> DataSet </text>
            <hint>
              Nome do objeto "TDataSet" que servira como fonte de dados
            </hint>
        </point>
        <point name="ListView">
            <text> ListView </text>
            <hint>
              Nome do objeto "TListView" que contera a lista de itens
            </hint>
        </point>
        <point name="Iterador">
            <text> i </text>
            <hint> Variavel de controle da iteracao </hint>
        </point>
        <code language="Delphi" delimiter="|"><![CDATA[]]></code>
    </template>
</codetemplate>
```

Listagem 5. Bloco de código do exemplo adicionado à tag Code

```
01 <code language="Delphi" delimiter="|"> <![CDATA[
02  {Abre o DataSet ou posiciona no primeiro registro}
03  if not |DataSet|.Active then
04   |*||DataSet|.Open
05  else
06   |*||DataSet|.First;
07   {Cria as colunas}
08   for |Iterador| := 0 to |DataSet|.Fields.Count -1 do begin
09    |*||NewColumn| := |ListView|.Columns.Add;
10    |*||NewColumn|.Caption := |DataSet|.Fields[|Iterador|].DisplayLabel;
11    |*||NewColumn|.AutoSize := True;
12   end;
13   {Efetua a iteracao nos registros}
14   while not |DataSet|.Eof do begin
15     |*|{Adiciona novo item}
16     |*||NewItem| := |ListView|.Items.Add;
17     |*|{Configura o "caption" do "item" com o primeiro campo}
18     |*|if |DataSet|.FieldCount > 0 then
19     |*||*||NewItem|.Caption := |DataSet|.Fields[0].AsString;
20     |*|{Cria os "subitens" com os outros campos}
21     |*|for |Iterador| := 1 to |DataSet|.Fields.Count -1 do
22     |*||*||NewItem|.SubItems.Add(|DataSet|.Fields[i].DisplayText);
23     |*||DataSet|.Next;
24   end;]]>
25 </code>
```

Conjunto de Characters

Reforçando a observação citada anteriormente, evite a utilização de characters acentuados. O interpretador não é unicode e erros podem ocorrer se usarmos characters acentuados.

Ponto de Código	text	hint
NewItem	NewItem	Nome do objeto “TListItem” que conterá as instâncias dos itens do ListView.
NewColumn	NewColumn	Nome do objeto “TListColumn” que conterá as colunas do ListView.
DataSet	DataSet	Nome do objeto “TDataSet” que servirá como fonte de dados.
ListView	ListView	Nome do objeto “TListView” que conterá a lista de itens.
Iterador	Iterador	Variável de controle da iteração.

Tabela 4. Pontos de código do novo Template

A tag *hint* dos pontos de código descreve a funcionalidade de cada ponto. Após a inserção dos pontos de código, o *Live Template* estará semelhante à **Listagem 4**.

Código

Agora será adicionado o código do novo *Live Template* (na *tag code*), conforme a **Listagem 5** (a numeração é apenas para fins de explicação do código).

Da linha 4 à linha 6 o código abre o *DataSet* ou posiciona o cursor no primeiro registro, caso o mesmo já esteja aberto. Logo, esse novo *template* somente funcionará com *DataSets* bi-direcionais, como o *SimpleDataSet* ou *ClientDataSet*.

Da linha 8 à linha 11 são criadas as colunas do *ListView* de acordo com os *Fields* do *DataSet*. A linha 14 inicia a iteração dos registros do *DataSet*. A linha 16 instancia um novo *ListItem*, enquanto na linha 19 adicionamos o valor do primeiro *Field* para a propriedade *Caption* do *ListItem*.

As linhas 21 e 22 adicionam os valores dos outros *Fields* às colunas do *ListView*. Além disso, serão adicionados comentários ao código para facilitar o entendimento de quem vier a utilizar o *template*.

Scripts

Para facilitar ainda mais a escrita desse código, o *template* efetuará a declaração automática das variáveis/objetos utilizados no mesmo. Para isso, será utilizado o evento *onleave* do *template*. Isso significa que, no momento da saída (acionando a tecla TAB), o script será executado (**Listagem 6**).

Listagem 6. Script para declaração automática de variáveis/objetos

```
<script language="Delphi" onenter="false" onleave="true">
    DeclareVariable(|NewItem|);
    DeclareVariable(|NewColumn|);
    DeclareVariable(|Iterador|);
</script>
```

Nesse ponto o *template* já está concluído (**Listagem 7**).

Exemplo de utilização

Vamos criar uma pequena aplicação do tipo *VCL Forms Application - Delphi for Win32* para utilizar o *template*. Salve-a como desejar. No formulário vazio adicione os componentes e altere suas propriedades de acordo com a **Tabela 5.**

Análise de Tipos

O BDS é capaz de analisar os objetos utilizados nos pontos de código e declarar os tipos das variáveis e/ou objetos corretamente. Para isso, é preciso que as units necessárias estejam adicionadas a cláusula *uses* (nesse exemplo, temos que ter a unit *ComCtrls*).

Livros de Delphi

Delphi - Implementação de Algoritmos e Técnicas p/ Ambientes Visuais
Visual Books
R$ 62,00

Delphi: para Todas as Versões
Ciência Moderna

R$ 29,00

Estudo Dirigido de Delphi 2005
Érica
R$ 78,50

Delphi 2005: Aplicações de Banco de Dados com InterBase 7.5 e MySQL 4.0.23
Érica

R$ 149,50

Introdução ao Desenvolvimento de Aplicações em Delphi
Uniderp
R$ 50,00

O Melhor da Delphi Line (CD 1 e 2)
Sistemas e Exemplos com fontes e Delphi
Ramos da Informática

10% desconto
De: R$ 134,00 Por: R$ 120,00

Delphi e Relatórios - Treinamento a Distancia (CD)2005
Clube Delphi
10% desconto
De: R$ 200,00 Por: R$ 180,00

Guia do Desenvolvedor de Delphi for .NET
Makron Books

10% desconto
De: R$ 173,50 Por: R$ 156,15

Delphi 8 Para Plataforma .NET: Curso Completo
Axcel
10% desconto
De: R$ 249,00 Por: R$ 224,10

Sistema Comercial Integrado com Delphi 2005: Cadastro e Estoque
Érica

10% desconto
De: R$ 135,00 Por: R$ 121,50

Frete Grátis
Para todo o Brasil!
3x no Visa e Mastercard
e 6x no Hipercard

Outras facilidades de pagamento:

livrosdeprogramacao
.com.br

Nome	Componente	Propriedade	Valor
TblVendors	Table	DatabaseName	DBDEMOS
		TableName	vendors.db
LstVendors	ListView	Align	alClient
		ViewStyle	vsReport
		RowSelect	True
		HotTrackStyles> htHandPoint	True
		HotTrackStyles> htUnderLineHot	True

Tabela 5. Componentes adicionados ao formulário e suas propriedades

Um último ajuste: aumente o tamanho do formulário alterando a largura para 800 *pixels* e a altura para 600 *pixels*. Para popular o *ListView* utilize o novo *template*, através do evento *OnCreate* ou *OnShow*, invocando a lista de *templates* através das teclas CTRL+J (**Figura 3**). Selecione *populalistview* e pressione ENTER.

Para finalizar, apenas preencha o ponto de código *DataSet* com o valor "TblVendors" e o ponto de código *ListView* com o valor "LstVendors" (pressionando a tecla TAB entre os pontos). Continue pressionando a tecla TAB até terminar os pontos de código, para que as variáveis sejam declaradas.

Compile o projeto e execute-o. A **Figura 4** mostra o formulário, listando todos os registros no *ListView*.

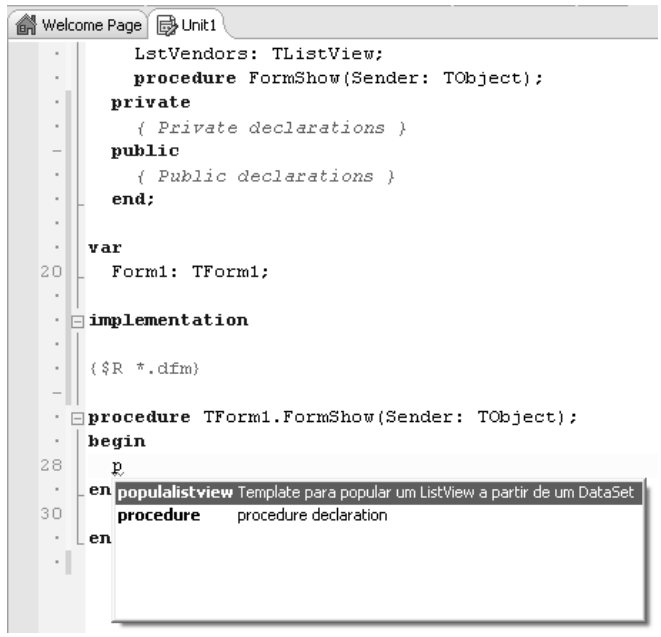
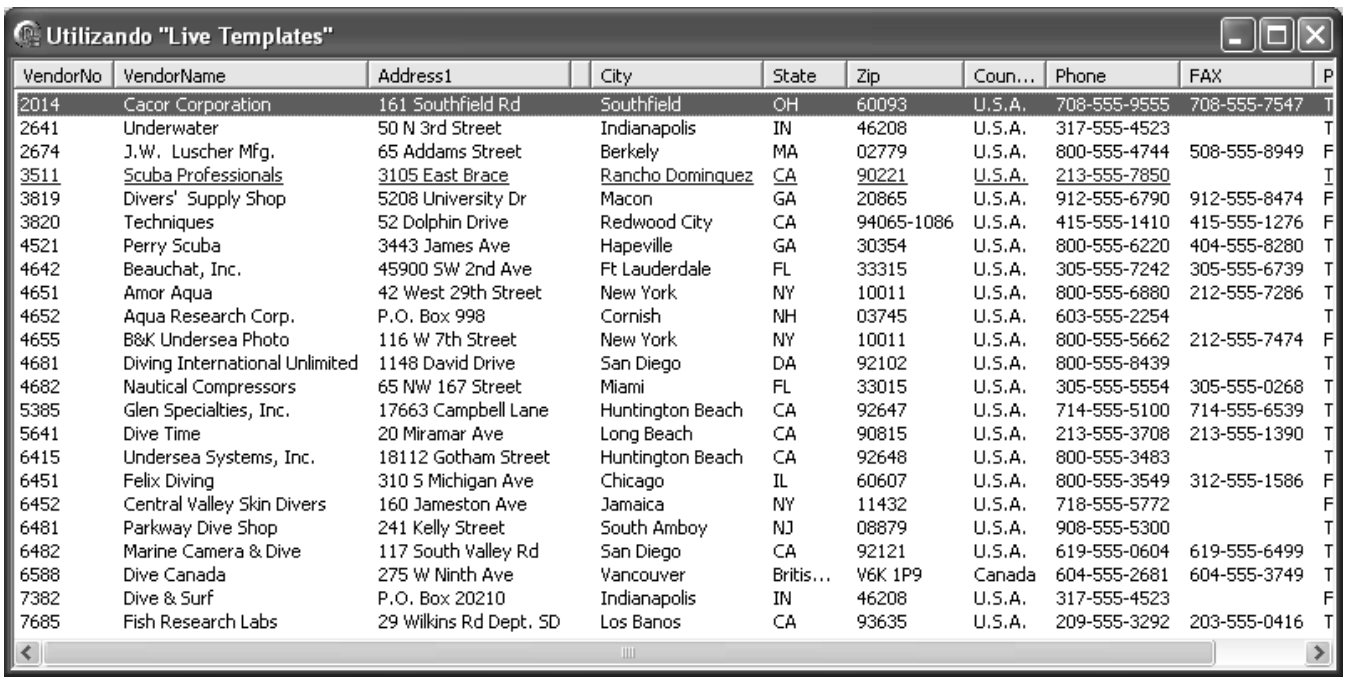


Figura 3. Acessando o novo Template no Delphi 2006

DBGrid x ListView

A principal diferença entre manipular um *DataSet* em um *DBGrid* e em um *ListView* é que, utilizando um *ListView*, cada registro é mantido como um objeto em memória (da classe *TListItem*). Para pequenos volumes de dados essa técnica pode ser bastante útil pelos recursos oferecidos por um objeto.



VendorNo	VendorName	Address1	City	State	Zip	Coun...	Phone	FAX	P
2014	Cacor Corporation	161 Southfield Rd	Southfield	OH	60093	U.S.A.	708-555-9555	708-555-7547	T
2641	Underwater	50 N 3rd Street	Indianapolis	IN	46208	U.S.A.	317-555-4523		T
2674	J.W. Luscher Mfg.	65 Addams Street	Berkely	MA	02779	U.S.A.	800-555-4744	508-555-8949	F
3511	Scuba Professionals	3105 East Brace	Rancho Dominquez	CA	90221	U.S.A.	213-555-7850		T
3819	Divers' Supply Shop	5208 University Dr	Macon	GA	20865	U.S.A.	912-555-6790	912-555-8474	F
3820	Techniques	52 Dolphin Drive	Redwood City	CA	94065-1086	U.S.A.	415-555-1410	415-555-1276	F
4521	Perry Scuba	3443 James Ave	Hapeville	GA	30354	U.S.A.	800-555-6220	404-555-8280	T
4642	Beauchat, Inc.	45900 SW 2nd Ave	Ft Lauderdale	FL	33315	U.S.A.	305-555-7242	305-555-6739	T
4651	Amor Aqua	42 West 29th Street	New York	NY	10011	U.S.A.	800-555-6880	212-555-7286	T
4652	Aqua Research Corp.	P.O. Box 998	Cornish	NH	03745	U.S.A.	603-555-2254		T
4655	B&K Undersea Photo	116 W 7th Street	New York	NY	10011	U.S.A.	800-555-5662	212-555-7474	F
4681	Diving International Unlimited	1148 David Drive	San Diego	DA	92102	U.S.A.	800-555-8439		T
4682	Nautical Compressors	65 NW 167 Street	Miami	FL	33015	U.S.A.	305-555-5554	305-555-0268	T
5385	Glen Specialties, Inc.	17663 Campbell Lane	Huntington Beach	CA	92647	U.S.A.	714-555-5100	714-555-6539	T
5641	Dive Time	20 Miramar Ave	Long Beach	CA	90815	U.S.A.	213-555-3708	213-555-1390	T
6415	Undersea Systems, Inc.	18112 Gotham Street	Huntington Beach	CA	92648	U.S.A.	800-555-3483		T
6451	Felix Diving	310 S Michigan Ave	Chicago	IL	60607	U.S.A.	800-555-3549	312-555-1586	F
6452	Central Valley Skin Divers	160 Jameston Ave	Jamaica	NY	11432	U.S.A.	718-555-5772		F
6481	Parkway Dive Shop	241 Kelly Street	South Amboy	NJ	08879	U.S.A.	908-555-5300		T
6482	Marine Camera & Dive	117 South Valley Rd	San Diego	CA	92121	U.S.A.	619-555-0604	619-555-6499	T
6588	Dive Canada	275 W Ninth Ave	Vancouver	Britis...	V6K 1P9	Canada	604-555-2681	604-555-3749	T
7382	Dive & Surf	P.O. Box 20210	Indianapolis	IN	46208	U.S.A.	317-555-4523		F
7685	Fish Research Labs	29 Wilkins Rd Dept. SD	Los Banos	CA	93635	U.S.A.	209-555-3292	203-555-0416	T

Figura 4. Resultado obtido com o código do novo Template

Listagem 7. Código completo do Live Template

```
<?xml version="1.0" encoding="utf-8" ?>
<codetemplate
 xmlns="http://schemas.borland.com/Delphi/2005/codetemplates"
 version="1.0.0">
    <template name="populalistview" invoke="manual">
        <description>
            Template para popular um ListView a partir de um DataSet
        </description>
        <author>
            Clube Delphi
        </author>
        <point name="NewItem">
            <text>
                NewItem
            </text>
            <hint>
                Nome do objeto "TListItem" que contera
                as instancias dos itens do "ListView"
            </hint>
        /point>
        <point name="NewColumn">
            <text>
                NewColumn
            </text>
            <hint>
                Nome do objeto "TListColumn" que contera
                as colunas do "ListView"
            </hint>
        </point>
        point name="DataSet">
            <text>
                DataSet
            </text>
            <hint>
                Nome do objeto "TDataSet" que servira como
                fonte de dados
            </hint>
        </point>
        <point name="ListView">
            <text>
                ListView
            </text>
            <hint>
                Nome do objeto "TListView" que contera a lista de itens
            </hint>
        </point>
        <point name="Iterador">
            <text>
                  i
            </text>
            <hint>
                Variavel de controle da iteracao
            </hint>
          </point>
        <script language="Delphi" onenter="false" onleave="true">
              DeclareVariable(|NewItem|);
              DeclareVariable(|NewColumn|);
              DeclareVariable(|Iterador|);
        </script>
        <code language="Delphi" delimiter="|"> <![CDATA[
        {Abre o DataSet ou posiciona no primeiro registro}
        if not |DataSet|.Active then
          |*||DataSet|.Open
        else
             |*||DataSet|.First;
             {Cria as colunas}
            for |Iterador| := 0 to |DataSet|.Fields.Count -1 do begin
             |*||NewColumn| := |ListView|.Columns.Add;
             |*||NewColumn|.Caption := |DataSet|.Fields[|Iterador|].DisplayLabel;
             |*||NewColumn|.AutoSize := True;
            end;
            {Efetua a iteracao nos registros}
            while not |DataSet|.Eof do begin
             |*|{Adiciona novo item}
             |*||NewItem| := |ListView|.Items.Add;
             |*|{Configura o "caption" do "item" com o primeiro campo}
             |*|if |DataSet|.FieldCount > 0 then
             |*||*||NewItem|.Caption := |DataSet|.Fields[0].AsString;
             |*|{Cria os "subitens" com os outros campos}
             |*|for |Iterador| := 1 to |DataSet|.Fields.Count -1 do
             |*||*||NewItem|.SubItems.Add(|DataSet|.Fields[i]. DisplayText);
             |*||DataSet|.Next;
            end;]]>
        </code>
    </template>
</codetemplate>
```

Conclusão

Como visto, os *Live Templates* adicionam muita produtividade na escrita de código. Além de possuir uma utilização simples, é possível criar nossos *templates*, maximizando ainda mais a produção de código. Com um pouco de criatividade é possível transformar códigos repetitivos em tarefas produtivas e eficazes, além de tornar os códigos um pouco mais padronizados entre diferentes desenvolvedores. ■

" O Maior Portal para Desenvolvedores da América Latina."

Acesse:

www.devmedia.com.br

DevMedia GROUP

Validações na prática

FABRÍCIO DESBESSEL
(fabricio.desbessel@terra.com.br) é professor de Linguagem de Programação do Curso Técnico em Informática do Colégio Frederico Jorge Logemann de Horizontina/RS e da FAHOR Faculdade Horizontina. Delphiano de coração está sempre disposto a provar que com o Delphi sempre teremos a melhor solução. Site www.fabricio.pro.br.

GABRIEL F. JANK
(gabrieljank@gmail.com) Delphiano há mais de cinco anos com experiência nas áreas de Web Services, Sistemas Distribuídos e Orientação a Objetos. Formado no Curso Técnico em Informática do Colégio Frederico Jorge Logemann de Horizontina/RS e Bacharelando em Sistemas de Informações pela SETREM de Três de Maio/RS.

O intuito deste artigo é demonstrar as várias formas de validação de dados, desde o BD até a interface, seja ela Windows ou Web, e os casos onde elas podem ser utilizadas em nossas aplicações. Mais especificamente apresentaremos validação no banco de dados, utilizando o InterBase/Firebird, validações em aplicações Win32 e validações em ASP.NET. Ao final de cada item, discutiremos as vantagens e desvantagens de cada um, para que você possa analisar como implementar as validações levando em conta cada projeto.

Tipos de validação no Banco de Dados

Validações em bancos de dados podem ser criadas através de *Constraints*, que permitem definir regras de integridade do banco e, por que não, regras de negócios. O IB/FB possui, nativamente, os seguintes tipos de regras:

- *Not Null*: cria uma regra simples informando que um campo é obrigatório, ou seja, não pode ser nulo. Essa regra é criada no momento de definição do campo;
- *Check Constraints*: cria uma regra, também à nível de campo, para validar o conteúdo recebido. Cada campo pode ter uma única regra *Check*, mas essa restrição pode ser superada com a utilização

CLUBEDELPHI PLUS!

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como validar informações em um servidor de aplicação DataSanp em aplicações multicamadas.

http://www.devmedia.com.br/articles/visualizacomponente2.asp?comp=2508

de um domínio que também contenha um *Check*. Na base de dados que criaremos, utilizaremos os *Checks* para impedir que seja informado um código menor ou igual a zero e também que os campos UF sejam informados em maiúsculo;

- *Primary Keys*: cria a chave primária da tabela e com isso também existirá uma regra que impedirá que registros diferentes possuam a mesma informação nesse campo;
- *Unique Key*: cria regras para não aceitar valores duplicados em um determinado campo, mesmo que o campo não seja chave primária. Na base de dados de exemplo, utilizaremos o *Unique* para o campo CPF, impedindo assim que os empregados possuam o mesmo CPF;
- *Foreign Keys*: cria regras de relações entre as tabelas impedindo que associações sejam feitas se os dados não constarem nas tabelas envolvidas.

Além dessas formas, que são as usadas com mais frequência, também podemos utilizar outros recursos do banco de dados para criar validações de preenchimento e validações envolvendo regras de negócio. Esses recursos são:

- *Trigger*: recurso que existe na maioria dos SGBDs. É uma implementação desenvolvida no banco, e que está atrelada a um determinado evento de uma tabela, como por exemplo, ao evento *Before Insert* da tabela EMPRESA. Muito parecido com os eventos dos objetos no Delphi;
- *Stored Procedure*: popularmente conhecido como *Procedimento*, e o nome já diz tudo, recebe parâmetros de entrada (opcionais), executa comandos e após o término, retorna os parâmetros de saída (opcionais) ou resultado;
- *Exception*: uma mensagem cadastrada na base de dados e que será utilizada nas *Triggers* e nas *Stored Procedures* para retornar erros para as aplicações clientes.

[1,1]

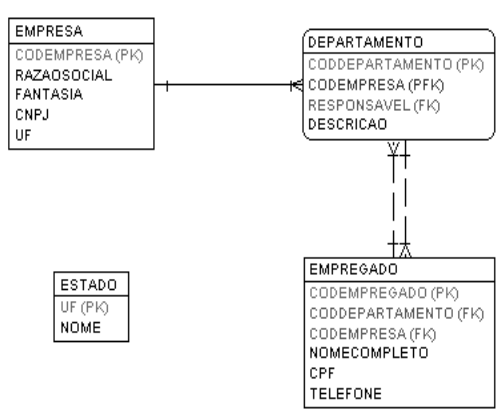


Figura 1. Diagrama ER da aplicação exemplo

Criando o banco de dados de exemplo

Para iniciar nosso exemplo prático, criaremos um banco de dados com alguns relacionamentos e campos interessantes para as técnicas de validação. Na **Figura 1** temos o DER da nossa base, onde usamos o Firebird 1.5.

A princípio parece estranho termos o ESTADO sem alguma relação, mesmo porque ela poderia estar relacionada com a tabela EMPRESA que também possui um campo UF. Mas justamente para exemplificar uma forma diferente de validação proposta neste artigo, vamos deixá-la assim, sem relacionamento.

Na **Listagem 1** temos o *script* para criação das quatro tabelas. Abra o seu gerenciador preferido (uma sugestão é o IBExpert - www.ibexpert.com), crie um novo banco de dados e execute o *script*. Nesse script criamos apenas as regras de validação para campos não nulos, através da declaração *Not Null*.

Listagem 1. Script de criação das entidades

```
CREATE TABLE "DEPARTAMENTO"
(
  "CODDEPARTAMENTO"INTEGER NOT NULL,
  "CODEMPRESA"    INTEGER NOT NULL,
  "RESPONSAVEL"INTEGER,
  "DESCRICAO" VARCHAR(60)
);

CREATE TABLE "EMPREGADO"
(
  "CODEMPREGADO"INTEGER NOT NULL,
  "CODDEPARTAMENTO"INTEGER NOT NULL,
  "CODEMPRESA"    INTEGER NOT NULL,
  "NOMECOMPLETO"VARCHAR(100),
  "CPF"      VARCHAR(11) NOT NULL,
  "TELEFONE"      VARCHAR(20)
);

CREATE TABLE "EMPRESA"
(
  "CODEMPRESA"    INTEGER NOT NULL,
  "RAZAOSOCIAL"VARCHAR(20),
  "FANTASIA"      VARCHAR(100),
  "CNPJ"      VARCHAR(14) NOT NULL,
  "UF"       CHAR(2)
);

CREATE TABLE "ESTADO"
(
  "UF"    CHAR(2) NOT NULL,
  "NOME"VARCHAR(20) NOT NULL
);
```

Primary Keys

Depois de criada as tabelas, criaremos as *Constraints* iniciando pelas *Primary Keys*, através do *script* da **Listagem 2**. É comum criar as *Primary Keys* junto com a tabela, informando o comando "Primary Key" ao lado do campo que se deseja como chave. Mas isso não é bom, pois o banco criará um nome genérico para essa *Constraint*, e com isso, dificultando um tratamento na aplicação cliente diferenciado para cada erro que ocorre.

Listagem 2. Script para criação das Primary keys

```
ALTER TABLE DEPARTAMENTO
ADD CONSTRAINT PK_DEPARTAMENTO
PRIMARY KEY (CODDEPARTAMENTO, CODEMPRESA);

ALTER TABLE EMPREGADO
ADD CONSTRAINT PK_EMPREGADO
PRIMARY KEY (CODEMPREGADO);

ALTER TABLE EMPRESA
ADD CONSTRAINT PK_EMPRESA
PRIMARY KEY (CODEMPRESA);

ALTER TABLE ESTADO
ADD CONSTRAINT PK_ESTADO
PRIMARY KEY (UF);
```

É importante lembrar que para um campo ser *Primary Key* ele não deve aceitar informação nula, claro. Ou seja, deve-se, no momento de criação do campo especificar-se *Not Null* para o campo.

Unique Key

Agora criaremos uma *Unique Key* para o campo CNPJ que, mesmo não sendo chave primária, não pode ter registros duplicados, utilizando o seguinte código:

```
ALTER TABLE EMPRESA
ADD CONSTRAINT UK_CNPJ UNIQUE (CNPJ);
```

Foreign Keys

Também criaremos as *Foreign Keys* para validar as relações entre as tabelas, impedindo que dados sejam informados sem constarem na tabela relacionada e impedindo que dados que estão sendo utilizados nas relações sejam excluídos da tabela (dependendo do valor especificado no *On Delete*). Utilize o *script* da **Listagem 3**.

Listagem 3. Script para criação das Foreign Keys

```
ALTER TABLE DEPARTAMENTO
ADD CONSTRAINT FK_CODEMPRESA
FOREIGN KEY (CODEMPRESA)
REFERENCES EMPRESA (CODEMPRESA)
ON UPDATE NO ACTION
ON DELETE NO ACTION;

ALTER TABLE DEPARTAMENTO
ADD CONSTRAINT FK_RESPONSAVEL
FOREIGN KEY (RESPONSAVEL)
REFERENCES EMPREGADO (CODEMPREGADO)
ON UPDATE NO ACTION
ON DELETE NO ACTION;

ALTER TABLE EMPREGADO
ADD CONSTRAINT FK_CODDEPARTAMENTO_CODEMPRESA
FOREIGN KEY (CODDEPARTAMENTO, CODEMPRESA)
REFERENCES DEPARTAMENTO (CODDEPARTAMENTO, CODEMPRESA)
ON UPDATE NO ACTION
ON DELETE NO ACTION;
```

Check

Está faltando ainda criar as validações do tipo *Check*. Execute o *script* da **Listagem 4** que impedirá que um campo "código" tenha um valor menor que 1, e que o campo UF seja informado em letras minúsculas.

Essas validações são básicas para os mais diversos bancos de dados, podendo ou não mudar um pouco a sintaxe.

Listagem 4. Script para criar os Check's

```
ALTER TABLE EMPREGADO
ADD CONSTRAINT CK_EMPREGADO
CHECK (CODEMPREGADO > 1);

ALTER TABLE EMPRESA
ADD CONSTRAINT CK_EMPRESA
CHECK (CODEMPRESA > 1);

ALTER TABLE EMPRESA
ADD CONSTRAINT CK_EMPRESA_UF
CHECK (UF = UPPER(UF));

ALTER TABLE DEPARTAMENTO
ADD CONSTRAINT CK_DEPARTAMENTO
CHECK (CODDEPARTAMENTO > 1);

ALTER TABLE ESTADO
ADD CONSTRAINT CK_ESTADO
CHECK (UF = UPPER(UF));
```

Validações avançadas no banco

Como a nossa base de dados está bem montada, com todas as relações e validações básicas de obrigatoriedade e de unicidade, e tendo em vista que nossas aplicações clientes receberão os erros caso alguma dessas regras não seja atendida, vamos nos focar em problemas mais complexos.

> **1º Problema:** Na tabela DEPARTAMENTO, temos um campo que se refere ao empregado responsável pelo departamento. Tendo em vista nossa base ser multi-empresa e multi-departamentos, vamos definir que somente pode ser atrelado a esse campo um empregado que faça parte desse departamento e não qualquer um da base de dados. Essa validação já é realizada pela chave-estrangeira que permitirá somente códigos de empregados cadastrados.

Para resolver esse problema, vamos criar duas *Triggers* no banco, referentes à tabela DEPARTAMENTO. Uma *Trigger* para o evento *Before Insert* que será executada antes da inserção real dos dados na base e no *Before Update* que se refere ao mesmo caso, mas que será executado antes da alteração dos dados. Além disso, criaremos duas *Exceptions* com as mensagens de erro.

Veja o *script* na **Listagem 5**.

Listagem 5. Script para solução do 1º Problema

```
CREATE EXCEPTION E_RESP_VAZIO
  'Durante a inclusão do departamento, nenhum responsável deve ser
informado!';

CREATE EXCEPTION E_RESP_INVALIDO
  'O responsável deve ser um empregado do departamento!';

Set Term ^ ;

CREATE TRIGGER DEPARTAMENTO_BI FOR DEPARTAMENTO
ACTIVE BEFORE INSERT POSITION 0
AS
begin
  if (NEW.responsavel IS NOT NULL) then
    exception e_resp_vazio;
end
^

CREATE TRIGGER DEPARTAMENTO_BU FOR DEPARTAMENTO
ACTIVE BEFORE UPDATE POSITION 0
AS
  declare variable v_CTD Integer;
begin
  if (NEW.responsavel IS NOT NULL) then
  begin
    select count(e.codempregado) from empregado e
    where e.coddepartamento = old.coddepartamento
    into v_CTD;

    if (v_CTD < 1) then
      exception e_resp_invalido;
     end
end
^
```

A *Trigger Departamento_BI*, executada no *Before Insert*, será responsável por verificar se durante a inclusão dos dados o campo empregado está nulo ou não. O campo empregado durante a inclusão do departamento deve estar vazio, porque não existem empregados relacionados ao "futuro" departamento.

A outra *Trigger* (*Departamento_BU*), executada no *Before Update*, segue outra linha, ela terá de verificar se durante a alteração do departamento, o responsável que será informado é do mesmo departamento que está sendo alterado.

2º Problema: Na tabela EMPREGADO existe o campo CPF que está definido como único na base e é de preenchimento obrigatório. Mas esse campo deve possuir um número de CPF válido durante a inclusão e a alteração do empregado.

Para esse caso, vamos criar um procedimento (*Stored Procedure*) no banco, que faça a validação de um CPF e posteriormente vamos implementar as *Triggers* de *Before Insert* e *Before Update* na tabela EMPREGADO. As *Trigger* utilizarão o procedimento de validação de CPF e uma exceção que também deve ser criada.

Como esse procedimento necessita de algumas funções inexistentes nativamente no Firebird, teremos de registrar UDFs (veja box) na base de dados, que são: *Strlen* que retorna o tamanho de uma *string*, *SubStrlen* que efetua a cópia de partes de uma *string* e o *Mod* que retorna o resto de uma divisão. Veja o *script* da **Listagem 6** para registrar as UDFs e criar a *Stored Procedure*.

Observe que a *Trigger* serve tanto para o evento de *Update* quando para o de *Insert* da tabela EMPREGADO.

UDF

User Defined function são bibliotecas externas ao banco de dados com funções criadas pelo desenvolvedor, que podem ser registradas e utilizadas pelo Firebird. Dentro do diretório de instalação do Firebird existe o arquivo */UDF/ib_udf.sql* com uma série de UDFs prontas e o código SQL para registro delas na base.

Listagem 6. Script para solução do 2º Problema

```
DECLARE EXTERNAL FUNCTION STRLEN
    CSTRING(32767)
RETURNS INTEGER BY VALUE
ENTRY_POINT 'IB_UDF_strlen' MODULE_NAME 'ib_udf';

DECLARE EXTERNAL FUNCTION SUBSTRLEN
    CSTRING(80),
    SMALLINT,
    SMALLINT
RETURNS CSTRING(80) FREE_IT
ENTRY_POINT 'IB_UDF_substrlen' MODULE_NAME 'ib_udf';

DECLARE EXTERNAL FUNCTION MOD
    INTEGER,
    INTEGER
RETURNS DOUBLE PRECISION BY VALUE
ENTRY_POINT 'IB_UDF_mod' MODULE_NAME 'ib_udf';

CREATE EXCEPTION E_CPF_INVALIDO
  'Número de CPF inválido! Preencha novamente!';

SET TERM ^ ;

CREATE PROCEDURE VALIDACPF (
    VALOR VARCHAR(11))
RETURNS (
    RESULTADO CHAR(1))
AS
DECLARE VARIABLE V1 INTEGER;
DECLARE VARIABLE V2 INTEGER;
DECLARE VARIABLE V3 INTEGER;
DECLARE VARIABLE V4 INTEGER;
DECLARE VARIABLE V5 INTEGER;
DECLARE VARIABLE V6 INTEGER;
DECLARE VARIABLE V_FAT INTEGER;
DECLARE VARIABLE C CHAR(2);
DECLARE VARIABLE D CHAR(2);
begin
    V1 = strlen(VALOR);
    V3 = V1-2; V2=1; V_FAT=2; V5=0;
    while (V2<=V1-2)do
    begin
      V5 = V5+cast(substrlen(
        VALOR,V3,1)as integer)*V_FAT;
      V2 = V2+1; V3=V3-1; V_FAT=V_FAT+1;
    end
    V5 = 11-(mod(V5,11));
    if (V5>=10) then
      V5=0; V2=3; V6=0; V4=V1-2;
    while (V2<=V1) do
    begin
      V6 = V6+cast(substrlen(VALOR,V4,1)as integer)*V2;
      V2 = V2+1; V4=V4-1;
    end
    V6 = V6+V5*2;
    V6 = 11-(mod(V6,11));
    if (V6>=10) then
      V6 = 0;
    C = cast(V5 as char)||cast(V6 as char);
    D = substrlen(VALOR,10,1)||substrlen(VALOR,11,1);
    if (C=D) then
      RESULTADO = 'X';
    else
      RESULTADO = ' ';
    suspend;
end
^

CREATE TRIGGER EMPREGADO_BIU FOR EMPREGADO
ACTIVE BEFORE INSERT OR UPDATE POSITION 0
AS
  DECLARE VARIABLE R_CPF CHAR(1);
begin
  if (new.CPF IS NOT NULL) then
  begin
    EXECUTE PROCEDURE validacpf NEW.CPF
      RETURNING_VALUES R_CPF;
    if (R_CPF = ' ') then
      exception e_cpf_invalido;
  end
end
^
```

O exemplo utilizado para a validação do CPF pode muito bem ser utilizado para a validação do CNPJ da empresa, por exemplo, ou qualquer outro campo que faça necessário uma função específica.

> **3º Problema:** O uso de integridade referencial (*Foreign Key*) em campos onde os valores se repetem constantemente pode causar perda de performance quando os dados são selecionados (*Selects*). Isso pode ser chamado de baixa granularidade e, é aconselhável que tenhamos cuidado para que não aconteça isso na base de dados. Um exemplo típico é a utilização de uma tabela de estados para o sistema onde, 90% dos registros usam o mesmo estado. Mesmo assim, é necessário validar a escolha do estado com um já existente na base. E então? Como resolver?

Por isso, não criamos um integridade referencial entre as tabelas ESTADO e EMPRESA e para resolver o problema vamos criar *Triggers* na tabela de EMPRESA que, ao inserir ou alterar algum dado, verificará se existe um estado com o UF informado.

Também criaremos uma *Trigger* para impedir que um ESTADO que esteja sendo utilizado seja apagado e, por último se houver alteração em um ESTADO, refletir na EMPRESA. Caso aconteça algum erro na validação será levantada uma exceção genérica, que também vamos criar, mas a mensagem correta será passada na *Trigger*. Na **Listagem 7** temos o *script* para esses objetos do banco.

Listagem 7. Script para solução do 3º Problema

```
CREATE EXCEPTION "E_ESTADO" 'Geral';
SET TERM ^ ;
CREATE TRIGGER "EMPRESA_BIU" FOR "EMPRESA"
ACTIVE BEFORE INSERT OR UPDATE POSITION 0
AS
begin
  if (not exists (select * from ESTADO e
     where e.UF = new.UF)) then
     exception E_Estado
     'O Estado ' || new.UF || ' não está cadastrado!';
end
 ^

CREATE TRIGGER "ESTADO_BU" FOR "ESTADO"
ACTIVE AFTER UPDATE POSITION 0
As
Begin
  if (old.UF <> new.UF) then
    update EMPRESA e set e.UF = new.UF
    where e.UF = old.UF;
end
 ^

CREATE TRIGGER "ESTADO_BD" FOR "ESTADO"
ACTIVE BEFORE DELETE POSITION 0
As
Begin
  if (exists(select * from EMPRESA e
    where e.UF = ESTADO.UF)) then
    exception E_ESTADO
    'O Estado ' || ESTADO.NOME ||
      ' está sendo utilizado e não pode ser deletado!';
end
 ^
```

Com essa solução criamos uma integridade que não afetará o desempenho do banco de dados.

VANTAGENS

A validação na base de dados é muito vantajosa em casos onde várias aplicações são desenvolvidas, em tecnologias e ferramentas distintas, para a mesma base (trabalhando com o mesmo arquivo do banco de dados). Isso porque na base você garante que independente de existir ou não uma validação na aplicação cliente, os dados serão validados. Também aumenta a segurança de seus dados protegendo-os de usuários com conhecimentos de banco de dados e que queiram inserir e alterar registros diretamente na base. Exemplo: ERP desenvolvido com o Delphi e com uma aplicação satélite web desenvolvida em PHP, por exemplo, para cotação e pedidos. O desempenho no processamento é muito mais rápido, pois o processamento será no servidor (máquina com potência superior) e não no cliente possibilitando assim, utilizar máquinas clientes mais leves.

DESVANTAGENS

Para qualquer alteração nas regras de validação, ou até mesmo a adição de uma nova validação, o script com as alterações deve ser executado em todas as bases de dados dificultando assim a manutenção nos sistemas. Outro ponto é que existem diferenças entre os SGBDs no que diz respeito a *Triggers* e Stored Procedures , pois as mesmas não seguem o padrão SQL ANSI, então se você pensa em ter uma aplicação multi-banco esse é um ponto que deve ser levado em consideração.

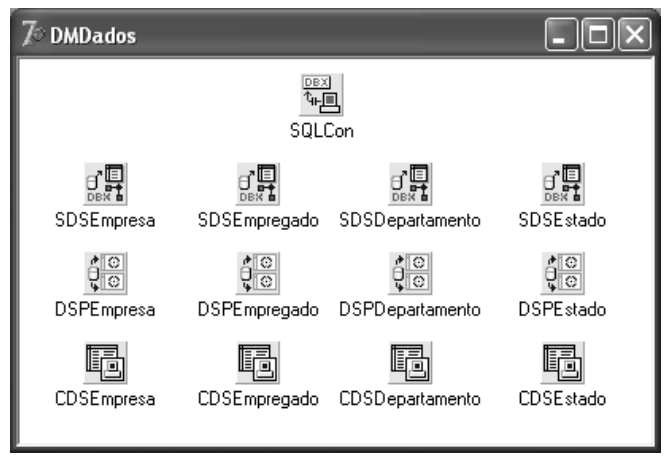


Figura 2. Data Module da aplicação

Tratando validação do banco na aplicação cliente

O banco de dados está pronto, contendo uma série de validações importantes além de integridades referenciais. Resta-nos criar uma aplicação cliente com acesso ao banco, para que possamos realizar os devidos testes.

Crie uma nova aplicação no Delphi e também um Data Module com o nome de "DMDados", que será o repositório dos componentes de conexão ao banco Firebird. Os componentes de acesso e suas respectivas configurações são:

- SQLConnection
  - *Name*: "SQLCon"
  - *ConnectionName*: "IBConnection"
  - *LoginPrompt*: *False*
  - *Params[SQLDialect]*: "3"
  - *Params[Database]*: <caminho do banco BASE.FDB>
- SQLDataSet
  - *Name*: "SDSEmpresa"
  - *SQLConnection*: *SQLCon*
  - *CommandType*: *ctTable*
  - *CommandText*: EMPRESA
- DataSetProvider
  - *Name*: "DSPEmpresa"
  - *DataSet*: *SDSEmpresa*
- ClientDataSet
  - *Name*: "CDSEmpresa"
  - *ProviderName*: *DSPEmpresa*

Observação: *Repita esse processo (com exceção é claro do SQLConnection) para as tabelas: EMPREGADO, DEPARTAMENTO e ESTADO.*

Veja na **Figura 2**, o layout do *DMDados*.

Vamos criar também no *DMDados* um evento único para o envio das alterações para a base de dados, e que será utilizado por todas as tabelas (*ClientDataSet*) nos eventos *AfterPost* e *AfterDelete*. Na **Listagem 8** o código do evento *GravaDados*.

Listagem 8. Evento para envio das alterações para a base

```
type
  TDMDados = class(TDataModule)
  ...
  procedure GravaDados(DataSet: TDataSet);
  private
...
implementation

{$R *.dfm}

procedure TDMDados.GravaDados(DataSet: TDataSet);
begin
  if DataSet.ClassType = TClientDataSet then
    (DataSet as TClientDataSet).ApplyUpdates(0);
end;
```

Vamos vincular os eventos *AfterPost* e *AfterDelete* de todos os componentes a esse método, para que todas as alterações realizadas em memória nos *ClientDataSets* sejam enviadas para o banco de dados. Na **Figura 3** vemos como vincular o método.

Feito isso, podemos criar as telas de cadastro, em nosso exemplo. Faremos a tela para o cadastro na tabela de EMPRESA, mas seguindo o mesmo padrão todas as outras telas podem ser feitas da mesma forma. No formulário principal, vamos montar uma tela semelhante a **Figura 4**, que consiste simplesmente em um *DBGrid*, um *DBNavigator* e *DBEdits*.

Rodando a aplicação pelo Delphi, e tentando inserir, por exemplo, um estado "XX" que não está cadastrado no banco, vamos receber uma mensagem de erro, semelhante a **Figura 5**.

Esse erro ocorreu devido à validação que está no banco de dados (comprove pelo texto da mensagem), onde não podemos inserir na tabela EMPRESA uma unidade de federação que não esteja previamente cadastrada. Mas existe algo interessante que ocorre nesse caso, é que a mensagem só é exibida quando rodamos a aplicação dentro do Delphi, pois estamos em tempo de depuração onde todos os erros são exibidos.

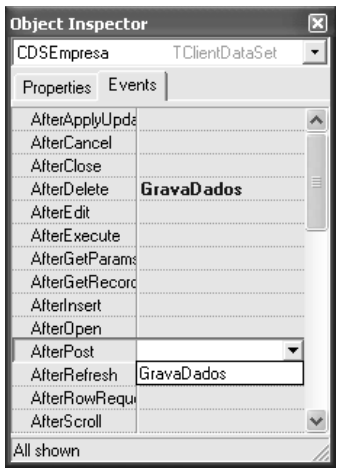


Figura 3. Vinculando os eventos

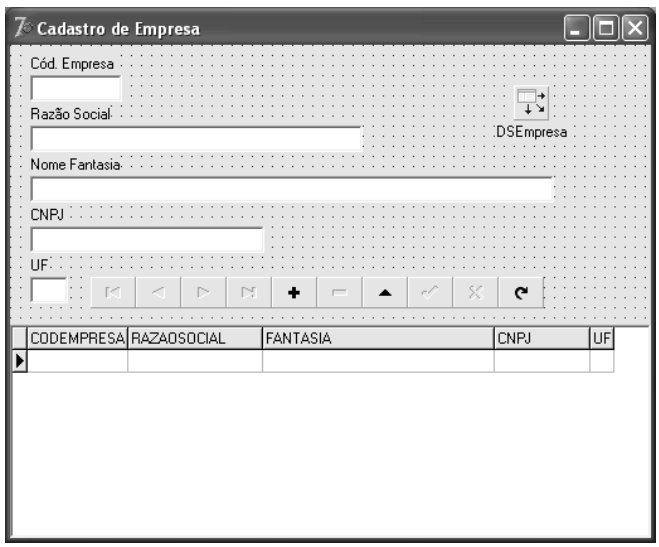


Figura 4. Tela de cadastro de empresa

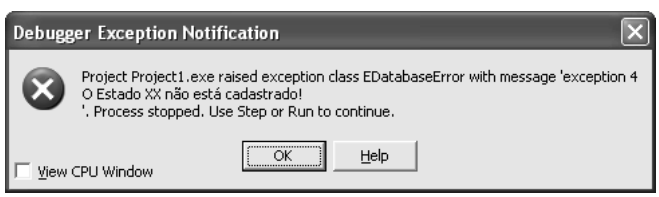


Figura 5. Mensagem de erro do Banco

Se rodarmos a aplicação fora do Delphi, ou sem o depurador ativado, não receberemos essa mensagem e nem o cadastramento do registro será bloqueado ou cancelado. Como resolver esse problema?

Usando o OnReconcileError

O nosso velho amigo *ClientDataSet* possui um evento chamado *OnReconcilerError* que é executado toda vez que uma exceção vem da base de dados (ou servidor de aplicação DataSnap em um arquitetura n-tier) e o registro não é corretamente aplicado. Nesse evento podemos fazer o tratamento do erro e reenviar os dados para a base, ou sim-

plesmente ignorar a inserção ou alteração dos registros.

No Delphi existe um formulário padrão para o tratamento completo nesse evento, e vamos utilizá-lo para demonstrar o que pode ser feito para tratar as exceções. No *DMDados*, selecione o CDSEMPRESA e vá até o evento *OnReconcileError*, dando um duplo clique para implementarmos o código do evento, conforme a **Listagem 9**.

Listagem 9. Evento para chamar o formulário de ReconcileError

```
procedure TDMDados.CDSEmpresaReconcileError(
  DataSet: TCustomClientDataSet;
  E: EReconcileError; UpdateKind: TUpdateKind;
  var Action: TReconcileAction);
begin
  Action := HandleReconcileError(DataSet, UpdateKind, E);
end;
```

É necessário também declarar no *uses*, a unit *RecError*, que é do formulário utilizado para tratar esse evento, lembrando que pode ser utilizado em todas as tabelas.

Observação: *Ao invés de utilizar o formulário padrão para tratar os erros provindos da base, você pode desenvolver sem problemas seu próprio método de controle, fazendo-se utilizar das variáveis disponíveis nesse evento, que são:*

DataSet: a tabela que possui dados inconsistentes, refere-se ao ClientDataSet;

E: variável com o erro. Possui a mensagem, número do erro, classe do erro etc.;

UpdateKind: define o método utilizado no momento do erro (Insert, Delete ou Update);

Action: ação que deve ser tomada. Exemplo: Cancelar, Reenviar, Abortar etc.

O resultado agora é que no momento da exceção, um formulário com a mensagem de erro é exibido, além disso, possibilita a correção das informações, o reenvio para a base ou o cancelamento do cadastro (**Figura 6**).

É possível traduzir esse formulário, e até mesmo alterar seu layout e suas opções. Os arquivos do formulário estão no diretório de instalação do Delphi 7 (pasta *Lib*) e podem ser copiados, para a pasta do projeto e adicionado a ele (**Figura 7**).

Validações na aplicação cliente

Após abordarmos as validações de dados vindas do banco, vamos ver a possibilidade de criarmos nossas validações na aplicação, método que consideramos o mais utilizado hoje pelos desenvolvedores no Delphi.

Existem várias maneiras de validar os dados, podemos focar em componentes simples de interação com o usuário como o *Edit*, por exemplo, ou podemos analisar os componentes de acesso a dados como o *ClientDataSet*. As validações junto aos componentes de acesso a dados será a pauta dessa parte do artigo.

Componentes de acesso a dados

Para os desenvolvedores que utilizam aplicações multicamadas (*3-tier*), é muito importante a validação de dados no

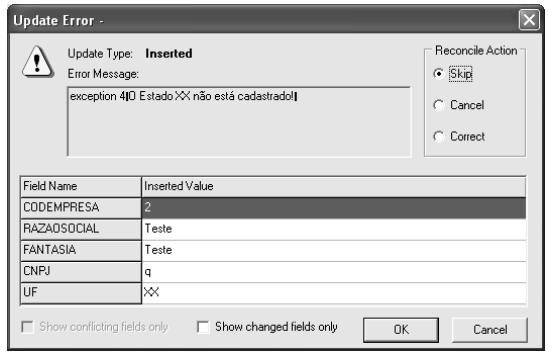


Figura 6. Tela com a exceção e opções de correção

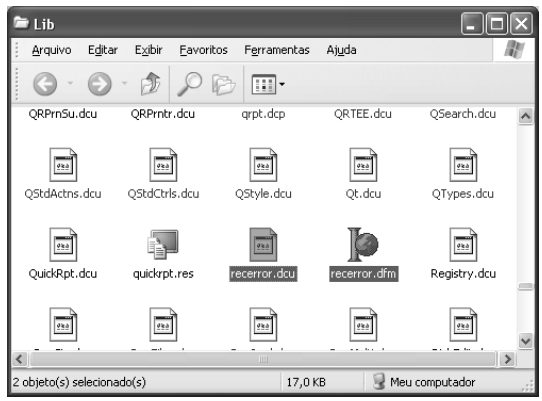


Figura 7. Arquivos do formulário do ReconcileError

lado da aplicação **servidora** e não no lado da aplicação cliente. Para esses tipos de validação podemos utilizar o seguinte:

- **Propriedade CustomConstraint (TField)**: comando *string* especificado no campo para sua validação, principalmente entre valores fixos e intervalos de dados. Exemplo: "X > 0 And X< 100";
- **Propriedade ConstraintErrorMessage (TField)**: mensagem que será mostrada, caso o valor estiver fora da validação definida na propriedade *CustomConstraint*;
- **Evento BeforeUpdateRecord (TDataSe Provider)**: evento que efetua a validação de todo o registro de dados antes que o mesmo seja enviado para a base de dados.

Dica: As propriedades citadas anteriormente em um exemplo multicamada, também podem ser utilizadas em aplicações cliente, pois as mesmas se aplicam a um *TField*.

Já do lado **cliente** da aplicação, tem outras opções para a validação de dados:

- **Evento OnValidate (TField)**: evento que efetua a validação do campo antes que o valor informado seja inserido no *buffer* do registro;

- **Propriedade Constraints (TDataSet)**: propriedade existente nos controles *DataSet*, em nosso exemplo no *ClientDataSet*, que funciona basicamente como um conjunto de regras entre os campos que podem ser definidas para a validação de dados.

Dica: Mesmo sendo casos muito parecidos, a configuração de regras na propriedade *Constraints* do *TDataSet* e a configuração individual no *TField* a partir da propriedade *CustomConstraint* possuem uma diferença bem significativa. A configuração no *TField* é individual e serve para que somente o valor daquele campo determinado seja validado, enquanto as regras definidas no *TDataSet* podem conter uma validação existente na relação entre diferentes campos.

Vamos continuar utilizando a aplicação de exemplo para criarmos as validações, e vamos focar apenas em um dos métodos citados anteriormente, que consideramos o mais flexível que é o *BeforeUpdateRecord.*

Vamos utilizar uma função simples de validação de CNPJ, que futuramente utilizaremos para validarmos o campo da tabela EMPRESA. Na **Listagem 11** temos a implementação que pode ser incluída no *DMDados*, logo após a cláusula *implementation*.

Listagem 10. Código da função para validar CNPJ

```
function ValidaCNPJ(const Value: string): boolean;
var
  i, soma, mult: integer;
begin
  ValidaCNPJ := false;
  if Length(Value) <> 14 then
    exit;
  soma := 0; mult := 2;
  for i := 12 downto 1 do
  begin
    soma := soma + StrToInt(Value[i]) * mult;
    mult := mult + 1;
    if mult > 9 then
      mult := 2;
  end;
  mult := soma mod 11;
  if mult <= 1 then
    mult := 0
  else
    mult := 11 - mult;
  if mult <> StrToInt(Value[13]) then
    exit;
  soma := 0;
  mult := 2;
  for i := 13 downto 1 do
  begin
    soma := soma + StrToInt(Value[i]) * mult;
    mult := mult + 1;
    if mult > 9 then
      mult := 2;
  end;
  mult := soma mod 11;
  if mult <= 1 then
    mult := 0
  else
    mult := 11 - mult;
  ValidaCNPJ := mult = StrToInt(Value[14]);
end;
```

Com a função disponível, vamos utilizar o evento *BeforeUpdateRecord* do *DataSetProvider (DSPEmpresa)* que, como citado anteriormente, é utilizado para validar os dados antes do envio do registro/atualização para a base de dados. Vamos iniciar, citando as principais características e recursos de validação desse evento:

- *SourceDS (TDataSet)*: é a referência ao *DataSet* em que está vinculado o *DataSetProvider*;
- *DeltaDS (TCustomClientDataSet)*: os dados do registro alterado;
- *UpdateKind (TUpdateKind)*: tipo de manutenção do registro (*ukModify*, *ukInsert* e *ukDelete*).

Vamos implementar o evento, fazendo a validação do CNPJ, e caso não seja válido, enviando uma mensagem de erro e solicitando a redigitação do mesmo (**Listagem 11**).

Listagem 11. Código do evento BeforeUpdateRecord

```
procedure TDMDados.DSPEmpresaBeforeUpdateRecord(
  Sender: TObject;
  SourceDS: TDataSet; DeltaDS: TCustomClientDataSet;
  UpdateKind: TUpdateKind; var Applied: Boolean);
begin
  if UpdateKind in [ukModify, ukInsert] then
  begin
    if (ValidaCNPJ(DeltaDS.FieldByName(
      'CNPJ').NewValue) = False) and (DeltaDS.
      FieldByName('CNPJ').NewValue <> '') then
        raise Exception.Create('Valor do Campo CNPJ '+
          'é inválido! Reinforme.');
  end;
end;
```

Como podemos ver, a validação será aplicada apenas em inserções e alterações no registro, além disso, chamamos a função de validação passando o novo valor do CNPJ (podemos fazer referência ao valor anterior utilizando *OldValue* ao invés de *NewValue*) para que ele seja validado, e em caso de erro utilizamos o *raise* para levantar uma exceção na aplicação cliente com nossa mensagem de erro.

VANTAGENS

As validações nas aplicações são práticas e de fácil desenvolvimento, diferente um pouco das validações no banco, onde devemos ter cuidados específicos para cada SGBD. Além disso, aplicando os conceitos de reutilização de nosso código, podemos centralizar nossas validações o que facilita a manutenção e distribuição das alterações.

As validações são muito utilizadas nas aplicações multicamadas no lado servidor, onde podemos definir as regras em um único local (de desenvolvimento e distribuição) sem necessidade de alterarmos nossas aplicações clientes e sem ficarmos presos à utilização de um único banco de dados.

DESVANTAGENS

Não podemos garantir que todo desenvolvedor fará as validações necessárias nos dados, diferente da validação na base de dados que "força" a validade e integridade dos dados. Além disso, em cenários complexos onde utilizamos diversas tecnologias e aplicações voltadas para uma mesma base, é trabalhoso manter e distribuir nossos códigos de validação em todas as aplicações.

Dica: É importante utilizarmos o *raise* ao invés do *ShowMessage* ou *MessageDlg* pois em aplicações multicamadas, necessitamos enviar a exceção para o cliente, caso contrário nosso erro acontecerá na aplicação servidora. Além disso, é uma má prática de programação usar referências a controles e rotinas visuais dentro de um DM.

Assim como o método *BeforeUpdateRecord*, poderíamos utilizar o evento *OnValidate* dos campos para validações mais customizadas, que necessitem chamadas de funções, cálculos entre outros.

Só lembrando que o evento *OnValidate* deve ser aplicado aos campos do *ClientDataSet* e não ao *SQLDataSet*, isso faz com que não possamos utilizar essa opção em aplicações servidoras onde as validações têm a vantagem de serem centralizadas.

Validações em ASP.NET

Na parte final deste artigo, vamos examinar como inserir mecanismos de validações em aplicações Web, usando o Delphi e ASP.NET.

O .NET Framework trouxe alguns componentes de validações para aplicações ASP.NET, que facilitam em muito nossa tarefa de validar campos. Veja a **Tabela 1** para conhecer os componentes e saber a função de cada uma.

Todos, menos o *ValidationSummary*, possuem a propriedade *ErrorMessage* para definição da mensagem de erro que aparecerá, caso o campo esteja preenchido incorretamente. Eles também possuem a propriedade *Text* onde você poderá escrever um texto alertando o usuário de como deverá preencher o campo. Esses componentes sempre deverão estar ligados, através da propriedade *ControlToValidate*, a um controle ASP.NET.

Componente	Função
RequiredFieldValidator	Para exigir o preenchimento de um campo.
CompareValidator	Para comparar o conteúdo de um campo. Pode-se comparar o conteúdo com tipos de dados como Inteiro, Double, Data e String.
RangeValidator	Para validar o preenchimento em uma faixa de valores.
RegularExpressionValidator	Para validar o preenchimento através de uma Expressão Regular (existem validações prontas para URLs, e-mails, telefones etc).
ValidationSummary	Agrupar todas as mensagens de erro das validações.
CustomValidator	Possibilita a criação de funções para validar.

Tabela 1. Componentes para validação em ASP.NET

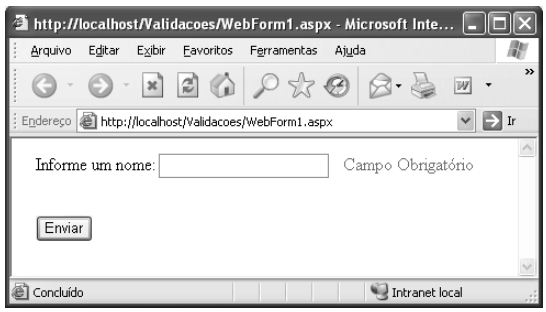


Figura 8. Validação para campo obrigatório em aplicações ASP.NET

Validando campos requeridos e em uma faixa de valores

Crie uma nova aplicação ASP.NET, no Delphi 8, 2005 ou 2006, através do menu *File>New>ASP.NET Web Application - Delphi for .NET*. Primeiramente vamos exigir o preenchimento de um campo (*TextBox*). Adicione um *Label* e na propriedade *Text* digite "Informe um nome:". Adicione um *TextBox* e um *RequiredFieldValidator*. Configure a propriedade *ErrorMessage* com uma mensagem solicitando o preenchimento do campo e a propriedade *ControlToValidate* para *TextBox1*.

Para testar a aplicação adicione um *Button* e mude a propriedade *Text* para "Enviar". Testando essa aplicação você poderá visualizar a validação em funcionamento (**Figura 8**).

O Delphi automaticamente utilizará funções *JavaScript* para validação no próprio cliente. Inclusive depois de aparecer a mensagem de erro e o usuário informar corretamente o campo, basta sair do campo e a mensagem desaparecerá.

Para testar o funcionamento do *RangeValidator* coloque um *Label* e defina a propriedade *Text* para "Informe um valor entre 1.000 e 2.000". Coloque ao lado um *TextBox* e depois um *RangeValidator*. Configure, no *RangeValidator* a propriedade *ControToValidate* para *TextBox2*, a propriedade *Type* para *Integer*, *MinimumValue* para "1000" e *MaximumValue* para "2000". Rode e teste a aplicação.

Comparando valores

Agora vamos utilizar um *CompareValidator*. Adicione um *Label* e defina a propriedade *Text* para "Informe uma Idade:". Ao lado coloque um *TextBox* e depois um *CompareValidator*. Configure a propriedade *ControlToValidate* para *TextBox3*, a propriedade *Operator* para *DataTypeCheck* e *Type* para *Integer*.

Com isso, estaremos validando um número inteiro. Com esse componente também podemos comparar valores preenchidos em *TextBoxes* diferentes. Vamos verificar isso colocando um outro *Label* e definindo sua propriedade *Text* para "Informe uma Senha:", ao lado coloque um *TextBox*, um outro *Label* e configure seu *Text* para "Confirme:" e um outro *TextBox*.

Vamos verificar se a senha e a confirmação são iguais. Adicione um *CompareValidator* e configure sua propriedade

Complete sua coleção!

Feita para Desenvolvedores de Software

SQL magazine

Escolha 03 edições e pague **R$ 6,90** em cada revista.

20

- Mapeamento Objeto Relacional. Aprenda a construir um mapeador O/R em .NET.

- Warehouse Builder. Construa uma data warehouse com a ferramenta ETL da Oracle-Partel

- UML. Aprenda passo a passo como construir um diagrama de sequências.

- Otimização. Influenciando o otimizador de consultas.

21

- Normalização. Aprenda a organizar melhor o esquema de seu banco de dados

- PostreSQL. Saiba o que são rules e como utilizá-las.

- SQL Server 2000. Entenda os modelos de recovery.

- InterBase 7.5. Quando e porque migrar.

- Particionamento do Oracle. Perigos e armadilhas Parte II.

22

- Backup. Saiba como proteger seus dados no Oracle e no SQL Server.

- UML. Conheça algumas opções de ferramentas livres - Parte II.

- Homologação de dados. Saiba como avaliar a qualidade de seu modelo de dados.

- Especial. Acessando servidores Oracle a partir do SQL Server 2000.

23

- Proteja-se. Saiba como funciona o SQL Injection e aprenda a lidar com ele no SQL Server.

- Administrando o Oracle. Conheça o Oracle Enterprise Manager.

- Backup. Saiba como efetuar o restore de Bds no SQL Server.

- Modelagem. Utilizando padrões na elaboração do projeto de banco de dados.

24

- Java + BD + Web. Trabalhe de forma integrada com: MySQL, Tomcat e Java com NetBeans.

- Oracle 10g. Aumente a disponibilidade de seu SGBD com o automatic Storage Management.

- PostgreSQL. Veja na prática como efetuar replicação entre servidores PostgreSQL.

25

- Clustering no SQL Server. Saiba para que serve e como funciona essa tecnologia de redundância.

- IBExpert. Configure e administre base de dados no Firebird e interbase.

- Caché. trabalhe com objetos diretamente no banco de dados.

- Oracle Discoverer. A prenda passo a passo a trabalhar com a ferramenta de análise de dados da Oracle.

26

- Mineração de dados. Entenda como ocorre a descoberta de informações em base de dados.

- IBExpert. Administrando o InterBase/Firebird. Conheça algumas características avançadas.

- UML. Ferramenta de modelagem free com engenharia reversa.

- MySQL. Transforme-se em um especialista na otimização de consultas - Parte II.

27

- MySQL. Veja como rodar um banco de dados MySQL em várias máquinas ao mesmo tempo!

- ER/Studio. Conheça esta excelente ferramenta para modelagem de dados.

- SQL. Descobrindo associações entre pares de linhas.

- SQL Server. Veja como enviar e-mails anexos no SQL Server sem o uso do SQL Mail.

28

- Dúvidas frequentes em bancos de dados.

- Desafio SQL. Construindo um modelo para um jogo de futebol on-line.

- Index Tuning Wizard. Como produzir tunings automáticos.

- Erwin conheça esta ferramenta de modelagem E/R.

29

- Oracle FLASHBACK. Recupere informações para um determinado ponto do tempo.

- SQL Server. Descubra se você esta trabalhando corretamente com o SQL Server.

- InterBase / Firebird. Trabalhando com stored procedures e triggers.

www.devmedia.com.br/msdn/anteriores

ou pela nossa central de atendimento: (21)2220-5375

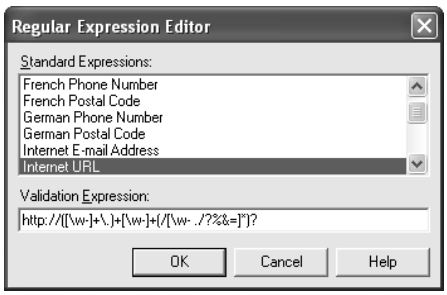


Figura 9. Expressão Regular para validação de URLs

ErrorMessage para "Senha e confirmação não conferem". Altere a propriedade *ControlToValidade* para *TextBox4*, *ControlToCompare* para *TextBox5*, *Operator* para *Equal* e *Type* para *String*. Rode a aplicação e teste as novas validações adicionadas.

Validação com expressões regulares

Por último, vamos verificar o preenchimento de um campo onde deverá ser utilizada uma *URL* válida. Para tanto vamos utilizar o *RegularExpressionValidator*. Coloque um *Label* e configure o *Text* para "Informe o endereço de um site:". Adicione um *TextBox* e um *RegularExpressionValidator*. Configure a propriedade *ControlToValidate* para *TextBox6* e na propriedade *ValidationExpression* clique no botão de reticências para selecionar o item *Internet URL* (**Figura 9**).

Validações personalizadas

Note que existem algumas expressões regulares prontas, mas você pode criar a sua. E, quando um desses componentes não valida o que queremos, podemos criar nossas próprias funções de validações, utilizando *JavaScript* ou *VBScript*.

Para demonstrar, vamos criar um código em *VBScript* para validar um número par. Acesse o código ASPX da aplicação e, entre as *tags* <*head*> e </*head*> coloque a função da **Listagem 12**.

Listagem 12. Função customizada de validação

```
<script language="vbscript">
   Sub ValidaCliente(source, arguments)
     If (arguments.Value mod 2) = 0 Then
 arguments.IsValid = true
     Else
 arguments.IsValid = false
     End If
   End Sub
</script>
```

Com a função já inserida, coloque um *Label* e configure o *Text* para "Informe um número par:". Adicione um *TextBox* e um *CustomValidator*. Configure a propriedade *ControlToValidate* para *TextBox7* e na propriedade *ClientValidationFunction* coloque "ValidaCliente" que é o nome da função de validação que será chamada pelo componente. Digite um texto para a mensagem, na propriedade *ErrorMessage*. Perceba que o nome da função deve ser informado sem o uso dos parênteses.

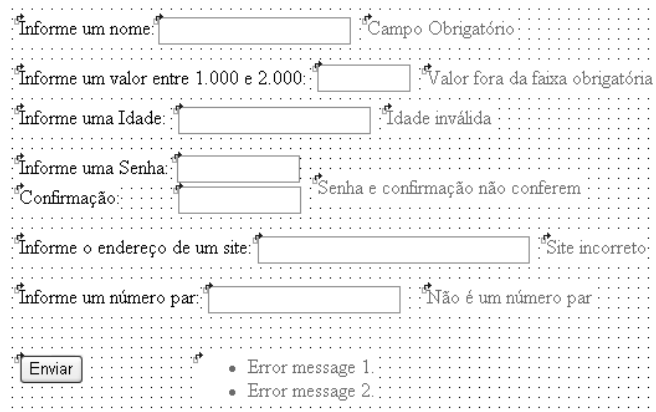


Figura 10. Layout da aplicação de validações em ASP.NET

VANTAGENS

Validações em aplicações ASP.NET são criadas com extrema facilidade e auxiliam no tráfego de dados com o servidor, uma vez que as validações são efetuadas no lado do cliente, ou seja, no browser, impedindo que dados errados sejam levados até o servidor e, com isso poupando o tempo do servidor retornar a validação incorreta.

DESVANTAGENS

Descentraliza a validação em aplicações com várias formas de acesso e, ainda requer uma segunda validação, pois as páginas podem ser salvas e alteradas excluindo-se as validações e enviando dados errados para o servidor.

Centralizando as mensagens

Uma opção interessante é juntar todas as mensagens de erros exibindo-as em um único local. Claro que, para isso, é necessário que as mensagens de erro configuradas na propriedade *ErrorMessage* esclareçam em qual campo aconteceu o erro. Basta adicionar o *ValidationSummary* no formulário para que as mensagens de erro sejam exibidos em um local específico.

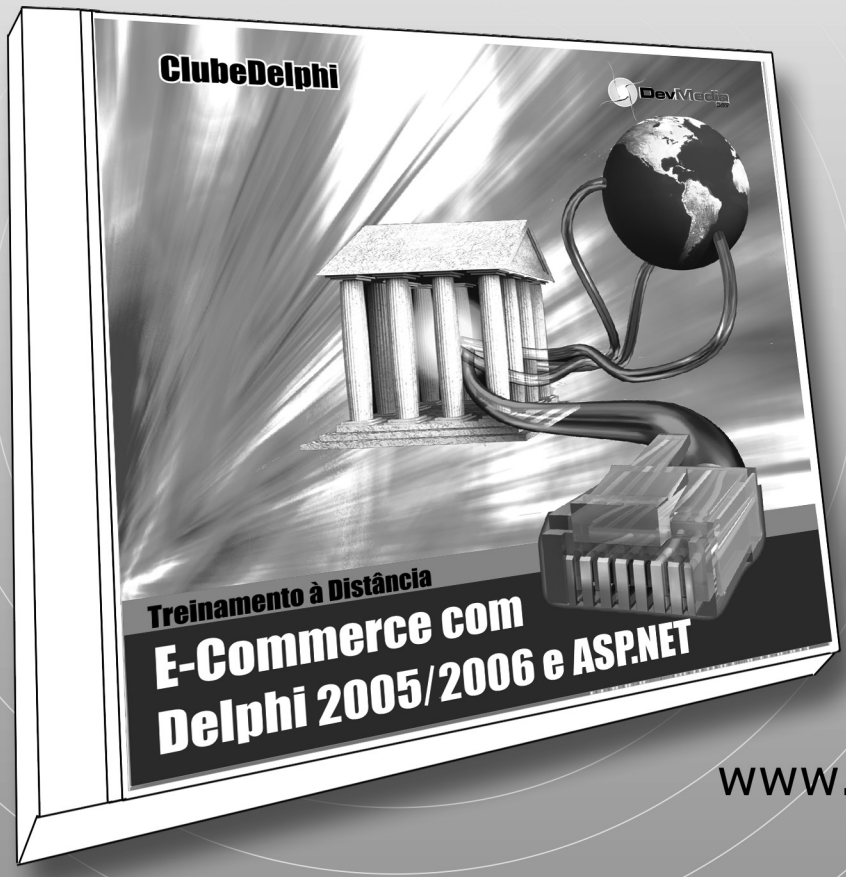
Se você deseja que as mensagens sejam exibidas somente nesse componente, selecione todos os componentes de validação e altere a propriedade *Display* para *None*. Se você quiser ainda poderá exibir os erros em uma mensagem de alerta. Para isso altere a propriedade *ShowSummary* para *False* e *ShowMessageBox* para *True*. Veja na **Figura 10** o layout da aplicação pronta.

Conclusão

A cada versão do Delphi temos novos componentes e formas de validar os campos utilizados em nossas aplicações. Com certeza essa atualização é fruto da real necessidade de protegermos nossa aplicação da má utilização pelos usuários. Nesse artigo vimos algumas técnicas. Defina a sua e comece a utilizar. ■

Treinamentos a distância.

Aprenda de forma prática, eficiente e econômica.



E-Commerce com Delphi 2005/2006 e ASP.NET

Aprenda a desenvolver poderosas aplicações Web através do Delphi 2005/2006 e as tecnologias ASP.NET, WebBroker, Intraweb e WebSnap.
O curso conta com módulos textuais;
35 horas de vídeo-aulas;
fórum exclusivo para os alunos e suporte on-line.

COMPRE JÁ O SEU!
www.devmedia.com.br/curso

Outros Cursos

Delphi e Relatórios

Com este curso você desenvolverá relatórios através do Delphi 7, do Delphi 2005 e Delphi 2006, utilizando as ferramentas Rave Reports, Quick Reports e Crystal Report. Além do módulo textual o curso contém mais de 20 horas de vídeo-aulas; fórum exclusivo e suporte

Evento à distância - Introdução ao .NET

Conheça os recursos do ASP.NET com o Visual Studio 2003 e aprenda a criar páginas ASP.NET; Webservices; trabalhar com a IDE do VS; componentização; DataGrid e criar aplicações de BD com o ADO.NET! São 6h de palestras em formato de vídeo-aulas e você pode tirar suas dúvidas com os palestrantes!

Evento à distância - Novidades do Delphi 2005

Conheça os recursos do Delphi 2005 e aprenda a criar páginas ASP.NET; Webservices; trabalhar com a nova IDE do Delphi; utilizar programação OO no Delphi 2005; trabalhar em equipe com o novo Starteam e criar aplicações de BD com o ADO.NET!

DevMedia

CRISTIANO CAETANO

Testando Web Services

CRISTIANO CAETANO
(c_caetano@hotmail.com)
autor do livro CVS: Controle de Versões e Desenvolvimento Colaborativo de Software. Com 10 anos de experiência, já trabalhou na área de qualidade e teste de software para grandes empresas como Zero G e DELL. Atualmente está exercendo a função de Software Test Engineer na HP Invent. O autor também pode ser contactado no seguinte endereço: spaces.msn.com/softwarequality.

Web Services é uma tecnologia baseada em XML e HTTP, cuja principal função é disponibilizar serviços interativos na Web que podem ser acessados (ou consumidos) por qualquer outra aplicação independente da linguagem ou plataforma em que a mesma foi construída.

O SOAP (*Simple Object Access Protocol*) é o padrão universal utilizado para a troca de mensagens entre as aplicações consumidoras e o Web Service. Para saber os métodos que o Web Services expõe, para que as aplicações possam consumir, utiliza-se um tipo de esquema XML chamado WSDL (*Web Services Description Language*).

SoapUI

SoapUI é um utilitário cuja principal função é prover um ambiente para consumir e testar Web Services. Todo o processo de criação e depuração dos testes é realizado numa interface visual e intuitiva. O *SoapUI* é um utilitário *open source* e distribuído sob a licença LGPL.

Pode ser obtido no site dos criadores (veja seção *Links*) ou diretamente no *SourceForge*. Entre as suas principais características, podemos destacar:

- Importação e geração automática das requisições descritas no WSDL;

CLUBEDELPHI PLUS!

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como criar Web Services no Delphi for .NET.

http://www.devmedia.com.br/articles/visualizacomponente2.asp?comp=511

- Capacidade de gerenciar um número ilimitado de requisições para cada operação;
- Gerenciamento de múltiplos *endpoints* para cada Web Service;
- Validação das requisições e respostas contra as suas definições no WSDL;
- Suporte para autenticação NTLM;
- Testes funcionais e de carga;
- Execução de testes em paralelo;
- Editores com *syntax highlight* e formatação automática;
- Suporta expressões XPATH;
- Suporta criação de testes complexos utilizando scripts *Groovy*.

Nota: Para trabalhar com a ferramenta, será necessário instalar a máquina virtual do Java (baixe em *www.java.com/getjava*). Após a instalação, execute o arquivo *soapui.bat* na pasta *bin* do diretório onde a ferramenta foi descompactada.

Preparando o WS para teste

Para fins de entendimento e para ser o mais didático possível, vamos analisar o *SoapUI* contra um Web Service simples rodando localmente no seu computador. O nosso Web Service foi criado com o Delphi 2006 e expõe apenas um método chamado *HelloEcho* cuja resposta sempre será a mesma *string* que foi enviada na requisição (passada como parâmetro).

Os passos para a criação de um Web Service estão fora do escopo deste artigo, no entanto você poderá obter maiores informações sobre esse assunto nas edições 55, 54, 37, 35, 34, 33, 32, 30, 28 e 27 da ClubeDelphi.

Para rodar o Web Service, que utilizaremos nas seções a seguir, você precisará instalar o IIS (*Internet Information Services*) e o .NET Framework 1.1. Por fim, basta criar um diretório virtual (dê o nome de "echoservice") no IIS e fazer o *deploy* do Web Service (apontando para o diretório que tem os fontes). Para confirmar se o Web Service está rodando corretamente, basta acessar o link "http://localhost/echoservice/WebService1.asmx" e, se tudo estiver certo, você verá uma página semelhante à exibida na **Figura 1.**

Criando o projeto de teste

Para obter o WSDL desse Web Service, basta acessar "http://localhost/echoservice/WebService1.asmx?WSDL". Para iniciar a utilização do *SoapUI*, você deverá criar um novo projeto. Para tal tarefa, clique no menu *File>New WSDL Project*. O *SoapUI* abrirá um diálogo solicitando o nome do novo projeto (**Figura 2**).

Você também precisará informar em qual pasta o projeto será salvo. Vale lembrar que durante modificação do seu projeto, o *SoapUI* se encarregará de salvar as modificações automaticamente. No entanto, se você desejar, poderá forçar o salvamento das modificações a qualquer momento, através do menu *File>Save all* ou pelo atalho CTRL+S. Vamos chamar o nosso projeto de "EchoService".Assim que o projeto for criado, ele aparecerá na árvore de projetos localizada no lado esquerdo do ambiente visual do *SoapUI*.

Definindo as interfaces (WSDL)

Em seguida devemos informar todas as interfaces expostas pelo Web Service que vamos consumir.

Para tal tarefa, devemos informar ao *SoapUI* a URL do WSDL contendo as definições de todas as interfaces expostas pelo Web Service, ou podemos apontar para um arquivo WSDL salvo localmente no seu computador.

Em resumo, você deverá realizar essa tarefa clicando com o botão direito do mouse sobre o nome do projeto localizado na árvore de projetos e escolher a opção desejada (**Figura 3**).

Lembre-se de confirmar a criação de requisições *default* para todas as operações quando for solicitado. A propósito, devemos esclarecer que o *SoapUI* suporta a edição de mais de um projeto ao mesmo tempo no seu ambiente visual.

Ao final do processo de importação do WSDL, o *SoapUI* incluirá um novo item abaixo do nome do projeto representando o WSDL importado. Clicando-se duas vezes sobre esse item você abrirá a janela *Interface Viewer*.

Figura 1. Web Service de exemplo rodando localmente no IIS

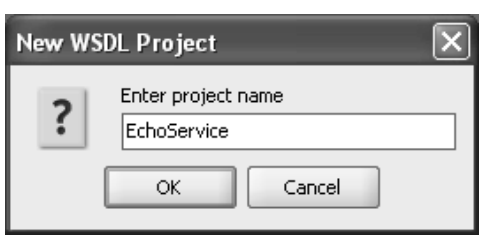


Figura 2. Criação de um novo projeto

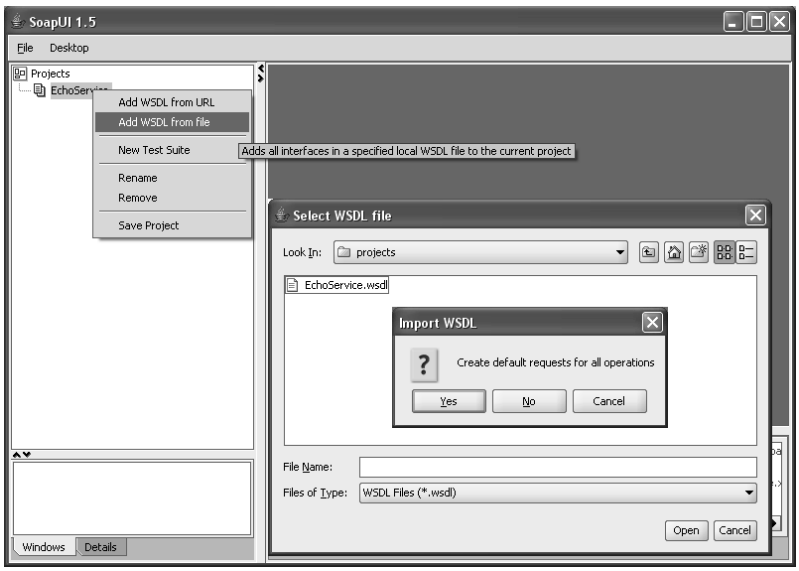


Figura 3. Importação das interfaces de um Web Service por meio de um arquivo WSDL

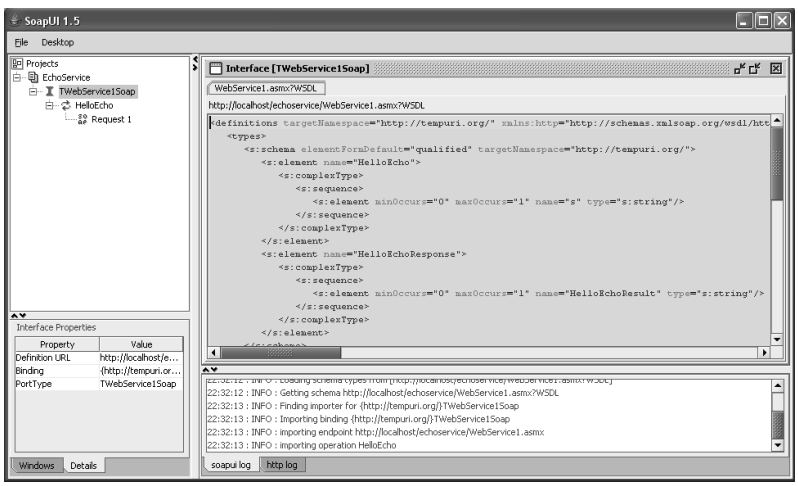


Figura 4. Interface Viewer

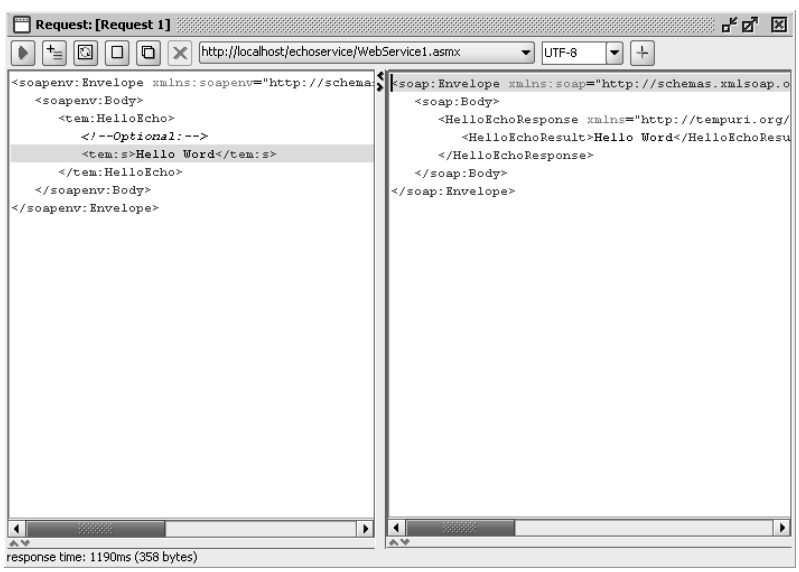


Figura 5. Request Editor: submetendo uma requisição ao Web Service

Através dela, você poderá visualizar o conteúdo do WSDL recém importado. Adicionalmente, você verá todas as operações expostas pelo Web Services, assim como, uma requisição *default* (*Request 1*) para cada operação, como pode ser visto na **Figura 4**.

Definindo as requisições

As requisições *default* são criadas para você testar o acesso ao Web Service e para facilitar a criação dos *Test Cases*. Você poderá testar o acesso ao Web Service clicando com o botão direito sobre a requisição *default* da operação que você desejar e escolher a opção *Open Request Editor*. No nosso cenário, você poderá abrir a requisição *default* (*Request 1*) da operação *HelloEcho*.

A janela *Request Editor* será aberta com lado esquerdo preenchido com a requisição SOAP que será enviada ao nosso Web Service. Você deverá preencher os campos da requisição conforme definido no WSDL.

Para consumir o único método exposto pelo nosso Web Service, devemos apenas fornecer uma *string* no único parâmetro disponível e submeter a requisição ao Web Service que está rodando localmente. A resposta da requisição é apresentada no lado direito do *Request Editor* (**Figura 5**).

Criando o Test Case

Quando você estiver seguro que o Web Service está se comportando adequadamente, você deverá criar um *Test Case* a partir dessa requisição que estiver manipulando. Para tal tarefa, você deverá clicar sobre a requisição desejada com o botão direito do mouse e escolher a opção *Add to TestCase*.

Você deverá informar o nome da *Test Suite*, o nome do *Test Case* e o nome do *Test Step* (no nosso exemplo, chamaremos de "Testes Básicos", "Test 1" e "Envia string Hello World", respectivamente). Uma *Test Suite* serve para agrupar um conjunto de *Test Cases* com características em comum e cada *Test Case* é formado por um ou mais *Test Steps*.

Além disso, devemos lembrar que cada *Test Step* representa o envio de uma requisição ao Web Service. Por fim, após a conclusão da criação do *Test Case*, o *SoapUI* incluirá um novo item abaixo do nome do projeto representando a *Test Suite* e o *Test Case*, respectivamente. Clicando-se duas vezes sobre o ícone do *Test Case*, você abrirá a janela *Test Suite Runner*.

Nessa janela, podemos iniciar a execução de todos os *Test Cases* que estiverem dentro da *Test Suite* (**Figura 6**).

Para editar ou depurar os *Test Steps*, dê um duplo clique sobre o nome do *Test Step*, abrindo a janela *Test Step Editor*. Por meio dessa janela, você poderá editar a requisição, inspecionar a resposta da requisição, assim como analisar o log HTTP gerado durante a execução dos testes, como pode ser visto na **Figura 7.**

Asserções

Ainda na janela *Test Step Editor*, você poderá definir asserções que serão executadas contra a resposta da requisição. Você poderá adicionar uma asserção clicando com o botão direito do mouse na aba *Assertions* localizada na parte inferior (**Figura 8**).

Atualmente, o *SoapUI* oferece cinco tipos diferentes de asserções para validar a resposta de uma requisição conforme a sua necessidade, como vemos na **Tabela 1**.

Asserção	Descrição
Schema Compliance	Valida as respostas das requisições contra o xml-schema definido no WSDL.
Simple Contains	Verifica a existência de uma string nas respostas das requisições.
Simple NotContains	Verifica a não existência de uma string nas respostas das requisições.
SOAP Fault	Verifica se a resposta da requisição não contém um soap-fault.
XPath Match	Compara o conteúdo das respostas das requisições contra uma expressão XPATH.

Tabela 1. Asserções fornecidas pelo SoapUI

Basicamente, as asserções são validações feitas para confirmar se a resposta de uma requisição contém as informações esperadas. Se as asserções falharem o *Test Step* e o *Test Case* indicarão um status de falha contendo uma descrição do que exatamente falhou (**Figura 9**).

Logs

À medida que se queira depurar *Test Cases* mais complexos e com uma grande quantidade de *Test Steps*, você deverá analisar os

Figura 6. Test Suite Runner executando todos os testes de uma Test Suite

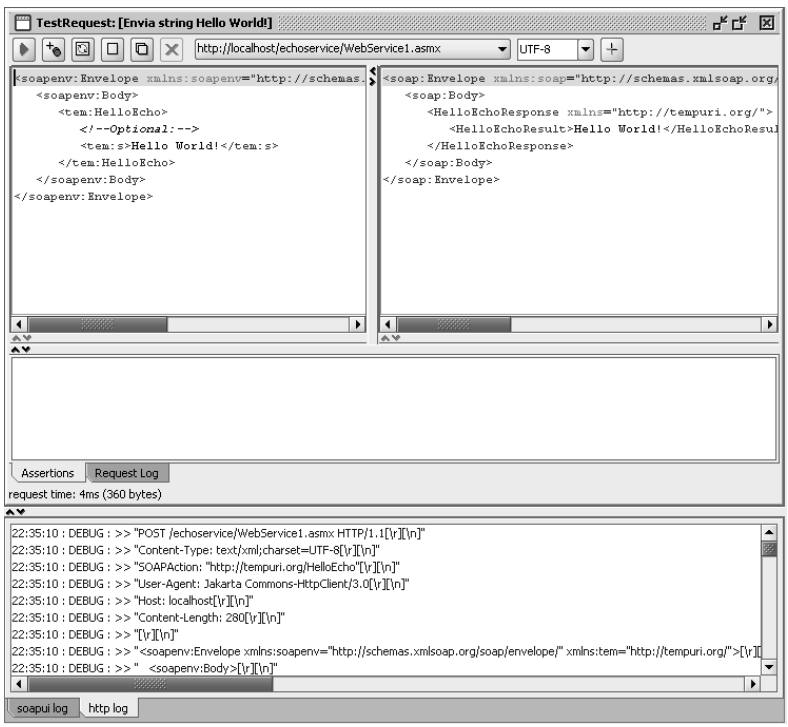


Figura 7. Test Step Editor para edição dos Test Steps

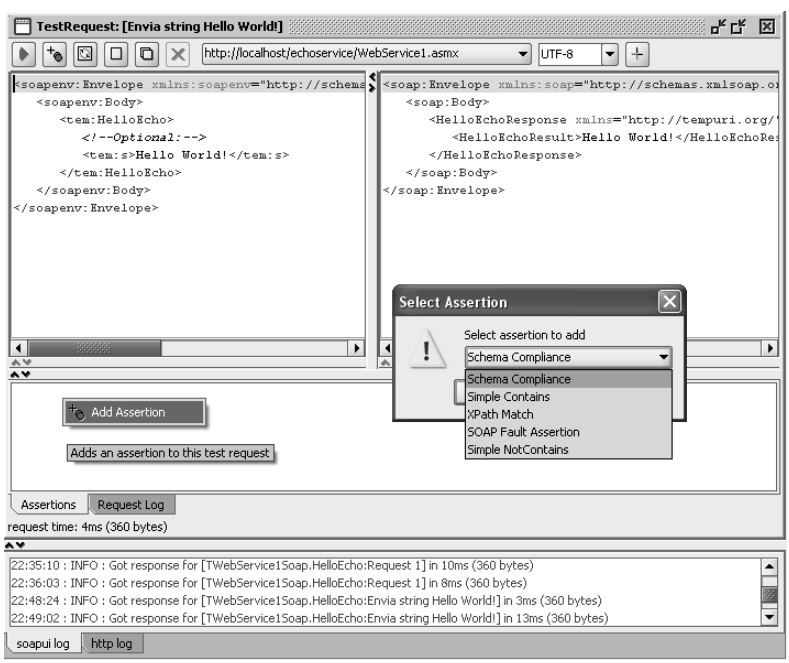


Figura 8. Test Step Editor adicionando asserções

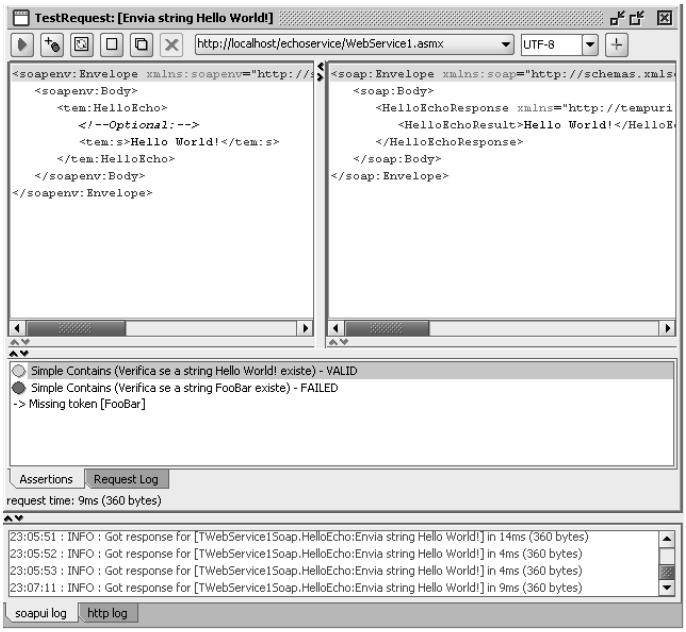


Figura 9. Asserções em ação

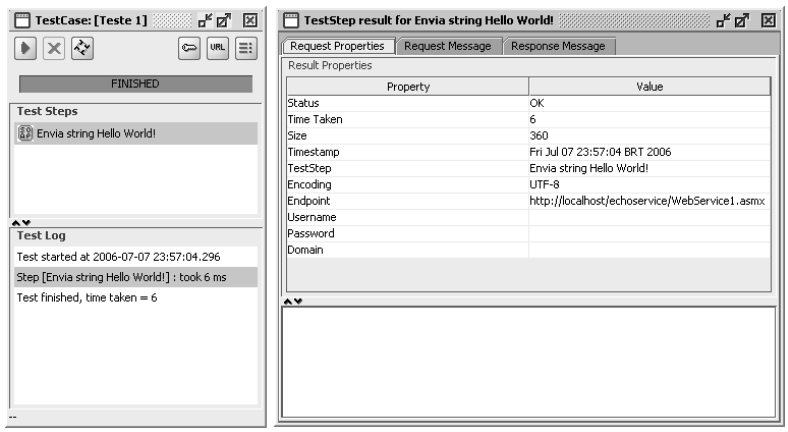


Figura 10. Depurando Test Cases complexos por meio do Test Log

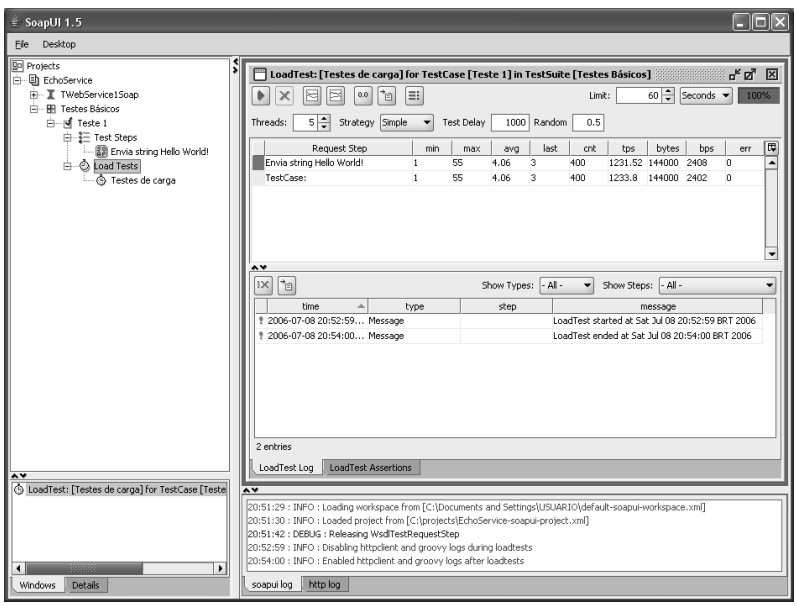


Figura 11. Testes de carga e performance

logs de execução dos *Test Steps*. Para tal tarefa, dê um duplo clique no *Test Step* desejado na seção *Test Log* localizada na parte inferior da janela *Test Suite Runner* (**Figura 10**).

Adicionando Test Steps

Conforme mencionamos anteriormente, cada *Test Step* é criado a partir de uma operação exposta pelo Web Service. No entanto, o *SoapUI* fornece outros tipos de *Test Steps* para ajudar a criar testes mais complexos e eficientes.

Para adicionar um novo *Test Step*, clique com o botão direito do mouse na seção *Test Steps* localizada na parte superior do *Test Step Editor* e selecione a opção *Insert Step*. Atualmente, o *SoapUI* oferece cinco tipos diferentes de *Test Steps* que podem ser adicionados ao *Test Case*, conforme descrito na **Tabela 2**.

Asserção	Descrição
Groovy Script	Linguagem de script baseada em Java para criar Test Cases e Asserções complexas.
Conditional Goto	Fornece a possibilidade de mudar o fluxo da condição da execução dos Test Steps de acordo com alguma condição.
Properties	Armazena propriedades que podem ser utilizadas entre os Test Steps como se fossem variáveis.
Delay	Permite a simulação de uma espera de acordo com um tempo determinado.
Property Transfer	Transfere dados entre os Test Steps.

Tabela 2. Test Steps fornecidos pelo SoapUI

Testes de carga e performance

Entre outras características, o *SoapUI* também fornece a possibilidade de executar testes de carga e performance contra os Web Services. Basicamente, a idéia é executar os *Test Cases* contra o Web Service segundo uma estratégia durante um período de tempo.

A estratégia para medir a performance pode incluir a configuração da quantidade de acessos simultâneos que serão realizados contra o Web Service, o período de tempo, entre outras configurações. Você também poderá criar asserções para identificar se o Web Service está atendendo determinado requisito de performance de acordo com as suas necessidades (**Figura 11**).

O *SoapUI* também apresenta os gráficos com as estatísticas da execução dos testes de carga e performance para facilitar a leitura e a análise dos dados (**Figura 12**).

Conclusão

Como vimos neste artigo, o *SoapUI* é uma ferramenta bastante eficiente e intuitiva. Em vez de perder várias horas debugando ou criando ferramentas temporárias para testar os seus Web Services, você poupará tempo criando executando os testes com os recursos fornecidos pelo *SoapUI* para garantir que os seus Web Services estão se comportando conforme o esperado e sem efeitos colaterais. Para obter maiores informações sobre o *SoapUI*, visite o site dos criadores, conforme endereço na seção *Links*. ■

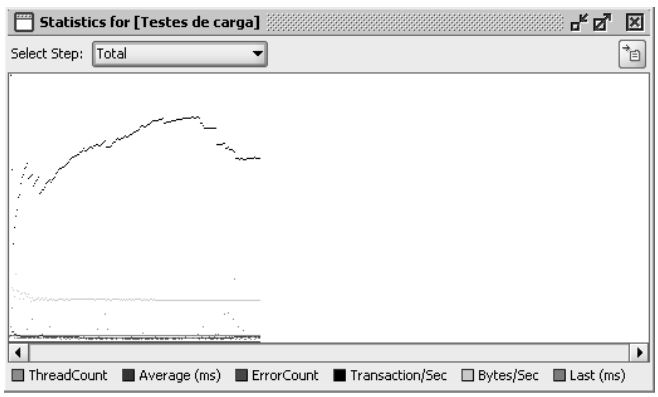


Figura 12. Gráfico com as estatísticas dos testes de carga e performance

Links

SoapUI Homepage

www.soapui.org

Groovy Home page

groovy.codehaus.org/Home

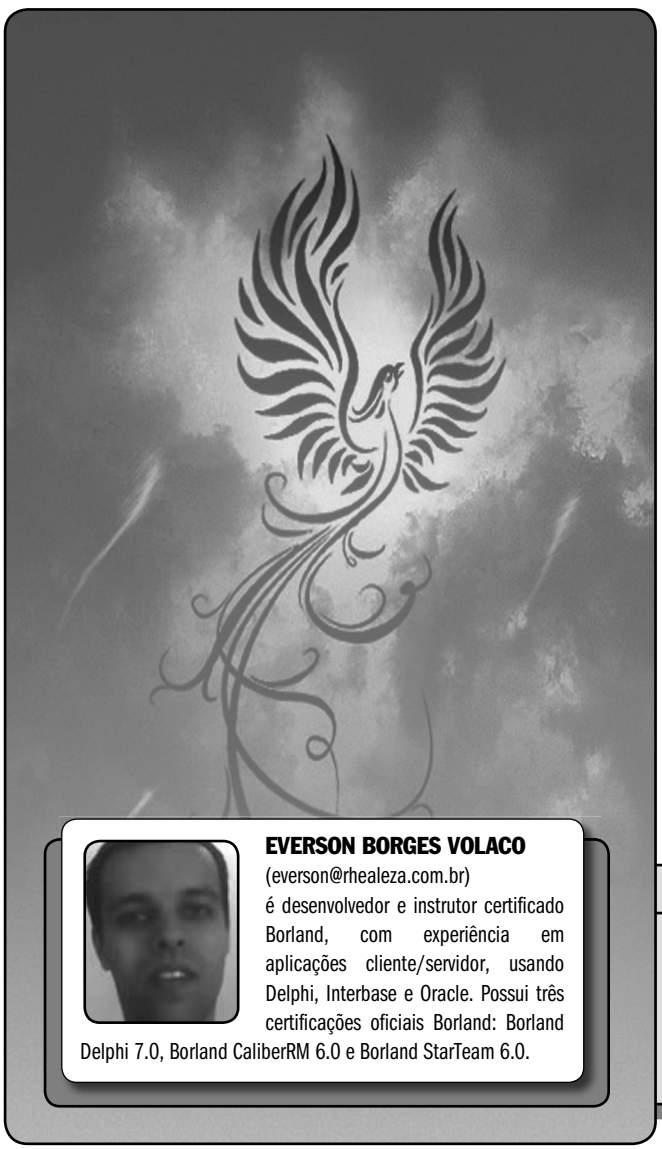
XPath (Tutorial)

www.w3schools.com/xpath

Wikipedia em português com informações sobre Web Services

pt.wikipedia.org/wiki/WebServices

Metadados no IB/FB



EVERSON BORGES VOLACO
(everson@rhealeza.com.br)
é desenvolvedor e instrutor certificado Borland, com experiência em aplicações cliente/servidor, usando Delphi, Interbase e Oracle. Possui três certificações oficiais Borland: Borland Delphi 7.0, Borland CaliberRM 6.0 e Borland StarTeam 6.0.

Neste artigo veremos como construir, utilizando o Delphi 7, uma ferramenta para criação de diagramas de ER (Entidade e Relacionamento) através da engenharia reversa das tabelas e *views* de bancos de dados InterBase e/ou Firebird .

O principal objetivo é demonstrar diversas técnicas que podem ser utilizadas nas mais variadas situações vividas pelos desenvolvedores no dia a dia. O *ERPlus*, nome dado à ferramenta, terá como principais funcionalidades:

- Permitir a seleção do banco de dados IB/FB pelo usuário;
- Gerar o digrama ER contendo todas as tabelas e *views* selecionadas pelo usuário;
- Listar para cada tabela ou *view* todos os seus campos com os respectivos tipos;
- Identificar as colunas definidas como chave primária e estrangeira;
- Mostrar os relacionamentos entre as tabelas (*Foreign Key*);
- Salvar os digramas em um formato XML;
- Abrir os diagramas salvos a partir do arquivo XML gerado;
- Exclusão de uma tabela ou *view* do diagrama;
- Arrastar as tabelas ou *views* dentro da área do diagrama;

CLUBEDELPHI PLUS!

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como acessar metadados no IB/FB a partir de aplicações .NET.

http://www.devmedia.com.br/articles/visualizacomponente2.asp?comp=2509

- Organização automática das tabelas e *views* contidas no diagrama.

Nota: Apesar de o aplicativo ter sido desenvolvido no Delphi 7, você pode compilá-lo sem problemas em qualquer outra versão do Delphi que possua suporte ao dbExpress, visto também que o mesmo não utiliza nenhum componente de terceiros.

Criando a aplicação

Crie uma nova aplicação no Delphi, altere o nome do formulário para "FrmPrincipal" e salve a unit como "untFrmPrincipal.pas". Para o arquivo de projeto salve-o como "ERPlus.dpr". Adicione alguns componentes visuais e não-visuais ao *FrmPrincipal* e configure-os como mostra a **Figura 1**.

Para o *mmPrincipal* (*MainMenu*) adicione os seguintes itens de menu:

- Diagrama
  - Novo
  - Abrir
  - Salvar
  - Salvar como
  - Fechar
- Banco de Dados
  - Configurar
- Tabelas
  - Selecionar Tabelas
  - Apagar Tabela Selecionada
- Outros
  - Organizar Layout
  - Visualizar Relacionamentos
  - Caixa Sobre
- Sair

Você pode utilizar o *imglPrincipal* (*ImageList*) para adicionar alguns ícones para os itens do menu principal e para os botões da *ToolBar*. Todas as ações dos menus e botões ficarão centralizados em *Actions* no *aclPrincipal* (*ActionList*).

Nota: Por questões de espaço explicarei neste artigo apenas os principais métodos da aplicação. Para acompanhar o passo a passo, você poderá baixar o código-fonte completo do *ERPlus* a partir do link para download da revista.

Crie uma nova unit e salve-a como "untTabela.pas". Para que possamos carregar o diagrama ER com as tabelas e *views* do banco de dados IB/FB, que será selecionado pelo usuário, vamos criar uma classe chamada "TTabela" na *unit* recém criada. Digite o código da **Listagem 1** na unit.

No código da listagem anterior criamos a classe *TTabela* descendente de *TPanel*. Para cada tabela ou *view* do banco de dados criaremos um objeto a partir dessa classe para mostrar os campos, com seus respectivos tipos, a(s) chave(s) primária(s) e a(s) chave(s) estrangeira(s). O atributo *FTabela* armazenará o nome da tabela lida do banco de dados.

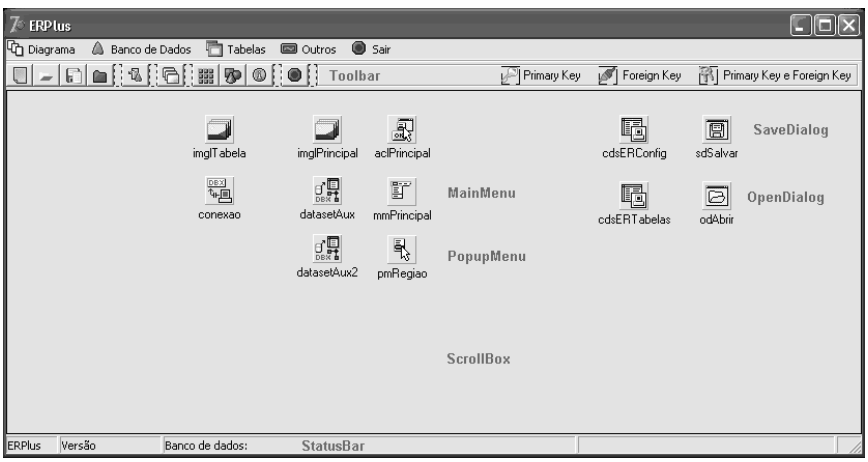


Figura 1. Formulário principal da aplicação em tempo de design

Figura 2. Formulário para configuração do banco de dados IB/FB

Utilizaremos o atributo *FPnlTitulo* para mostrar o nome da tabela como título dentro do diagrama. O atributo *FLVCampos*, que é do tipo *ListView*, será responsável em armazenar os campos e os tipos carregados da tabela do banco de dados em questão.

E por fim, o atributo *FChavesEstrangeiras* armazenará as tabelas aonde a tabela em questão possui relacionamentos, caso existam. No construtor da classe criamos e definimos os atributos da mesma. Adicione um novo formulário, altere seu nome para "FrmBD" e salve sua unit como "untFrmBD.pas". Adicione alguns componentes visuais e não-visuais (**Figura 2**).

Selecione o botão "..." (*sbCaminho*) e adicione o seguinte código ao seu evento *OnClick*:

```
if (OpenDialog1.Execute) and (
  OpenDialog1.FileName <> '') then
  edtCaminho.Text := OpenDialog1.FileName;
```

Crie um método chamado *validaCampos* e implemente-o conforme a **Listagem 2**.

Listagem 1. Criando a classe para acessar as tabelas e views do banco

```
unit untTabela;
interface
uses
  Classes, SysUtils, Graphics, Controls, ExtCtrls,
  ComCtrls;

type

  TTabela = class(TPanel)
    private
      FTabela: string;
      FPnlTitulo: TPanel;
      FLVCampos: TListView;
      FChavesEstrangeiras : TStringList;
      procedure EndDrag(Sender, Target: TObject;
        X, Y: Integer);
      procedure setTabela(const Value: string);
    public
      property Tabela: string read FTabela
        write setTabela;
      property PnlTitulo: TPanel read FPnlTitulo
        write FPnlTitulo;
      property Campos: TListView read FLVCampos
        write FLVCampos;
      property ChavesEstrangeiras: TStringList
        read FChavesEstrangeiras
        write FChavesEstrangeiras;
      constructor Create(AOwner: TComponent); override;
      destructor Destroy; override;
  end;

implementation

{ TTabela }

constructor TTabela.Create(AOwner: TComponent);
var
  Coluna: TListColumn;
begin
  inherited;
  Self.Height := 200;
  Self.Width := 240;
  Self.Ctl3D := False;
  Self.BevelInner := bvRaised;
  Self.BevelOuter := bvLowered;
  Self.TabStop := False;
  Self.OnEndDrag := EndDrag;
  FPnlTitulo := TPanel.Create(Self);
  FPnlTitulo.Parent := Self;
  FPnlTitulo.Caption := '';
  FPnlTitulo.Color := clMoneyGreen;
  FPnlTitulo.Align := alTop;
  FPnlTitulo.BevelInner := bvRaised;
  FPnlTitulo.BevelOuter := bvLowered;
  FPnlTitulo.TabStop := False;
  FPnlTitulo.Height := 15;
  FLVCampos := TListView.Create(Self);
  FLVCampos.Parent := Self;
  FLVCampos.Align := alClient;
  FLVCampos.ColumnClick := False;
  FLVCampos.GridLines := True;
  FLVCampos.ReadOnly := True;
  FLVCampos.TabStop := False;
  FLVCampos.ViewStyle := vsReport;
  Coluna := FLVCampos.Columns.Add;
  Coluna.Caption := 'Campo';
  Coluna.Width := 130;
  Coluna := FLVCampos.Columns.Add;
  Coluna.Caption := 'Tipo';
  Coluna.Width := 100;
  FChavesEstrangeiras := TStringList.Create;
end;

destructor TTabela.Destroy;
begin
  FreeAndNil(FPnlTitulo);
  FreeAndNil(FLVCampos);
  FreeAndNil(FChavesEstrangeiras);
  inherited;
end;

procedure TTabela.EndDrag(Sender, Target: TObject;
  X, Y: Integer);
begin
  Self.BevelWidth := 1;
end;

procedure TTabela.setTabela;
begin
  Self.FTabela := Value;
  if Assigned(FPnlTitulo) then
    FPnlTitulo.Caption := FTabela;
end;
end.
```

Listagem 2. Função validaCampos

```
function TFrmBD.validaCampos: Boolean;
begin
  Result := True;
  if edtCaminho.Text = '' then
  begin
    MessageDlg('Informe o caminho do arquivo de '+
      'banco de dados!', mtInformation, [mbOk], 0);
    edtCaminho.SetFocus;
    Result := False;
    Exit;
  end;
  if edtUsuario.Text = '' then
  begin
    MessageDlg('Informe o usuário!', mtInformation,
      [mbOk], 0);
    edtUsuario.SetFocus;
    Result := False;
    Exit;
  end;
  if edtSenha.Text = '' then
  begin
    MessageDlg('Informe a senha!', mtInformation,
      [mbOk], 0);
    edtSenha.SetFocus;
    Result := False;
    Exit;
  end;
end;
```

Para o botão *Testar* digite o código da **Listagem 3** em seu evento *OnClick*.

Listagem 3. Código para testar o usuário e senha do banco

```
uses untFrmPrincipal;
...
if not validaCampos then
  Abort;
Screen.Cursor := crHourGlass;
with FrmPrincipal.conexao do
begin
  if Connected then
    Connected := False;
  Params.Values['Database'] := edtCaminho.Text;
  Params.Values['User_Name'] := edtUsuario.Text;
  Params.Values['Password'] := edtSenha.Text;
end;
try
  FrmPrincipal.Conexao.Connected := True;
  Screen.Cursor := crDefault;
  if (Sender is TSpeedButton) then
    MessageDlg('Conectado com sucesso!', mtInformation, [mbOK], 0);
except
  on E : Exception do
  begin
    Screen.Cursor := crDefault;
    MessageDlg('Erro ao tentar conectar no '+
      'banco de dados!' + #13 + 'Mensagem Original: '+
      E.Message, mtError, [mbOk], 0);
    Abort;
  end;
end;
```

Selecione o botão *Cancelar* e altere sua propriedade *Kind* para *bkCancel*. Para o evento *OnClick* do botão OK digite:

```
sbTestar.OnClick(Sender);
ModalResult := mrOk;
```

Adicione mais um formulário a aplicação e altere seu nome para "FrmTabelas". Salve sua unit como "untFrmTabelas.pas". Adicione alguns componentes como mostra a **Figura 3**.

Selecione o botão OK e altere sua propriedade *Kind* para *bkOK*, no botão *Cancelar* altere para *bkCancel*. Utilizaremos o *ListBox* para carregar a lista de tabelas e *views* do banco de dados selecionado pelo usuário. Adicione mais um formulário a aplicação e altere seu nome para "FrmRegiao", salvando sua unit como "untFrmRegiao.pas".

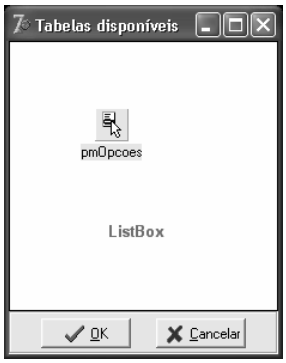


Figura 3. Formulário para seleção das tabelas e/ou views a serem adicionadas ao diagrama

Não adicionaremos nenhum componente a esse formulário, pois o mesmo será utilizado para mostrar o digrama ER. Mostraremos o formulário em tempo de execução dentro do *ScrollBox* presente no *FrmPrincipal*. No *FrmRegiao* adicione as seguintes variáveis em sua seção *private*:

```
private
  numLigacoes: Integer;
  vCanvas: array of TCanvas;
```

Utilizaremos um *array* de *TCanvas* para armazenar todos os relacionamentos que serão criados entre as tabelas do diagrama. Crie um método chamado *MontaGrade* e declare o mesmo dentro da seção *public* do *FrmRegiao*. Veja a implementação do método na **Listagem 4**.

Listagem 4. Método MontaGrade

```
procedure TFrmRegiao.MontaGrade;
var
  i: Integer;
begin
  if Color <> clWhite then
    Exit;
  i := 0;
  Canvas.Pen.Color := clAqua;
  while i < Width do
  begin
    Canvas.MoveTo(i, 0);
    Canvas.LineTo(i, Height);
    i := i + 20;
  end;
  i := 0;
  while i < Height do
  begin
    Canvas.MoveTo(0, i);
    Canvas.LineTo(Width, i);
    i := i + 20;
  end;
end;
```

Esse método será responsável em criar, através da propriedade *Canvas*, um grid verde dentro da área do *FrmRegiao*. Nessa área serão adicionadas as tabelas e *views* do diagrama. Crie mais um método público chamado *CriarLigacao* no *Frm Regiao*. Implemente-o com o código da **Listagem 5**.

O método *CriarLigacao* será responsável por desenhar o relacionamento entre as tabelas adicionadas no diagrama, através da classe *TCanvas*. Crie um método chamado "tabelaOrigemAdicionada" e implemente-o conforme a **Listagem 6.**

Listagem 5. Método para criar a ligação entre as tabelas

```
procedure TFrmRegiao.CriarLigacao(
  const tabelaDestino: TTabela);
var
  i: Integer;
  tabelaOrigem: TTabela;
begin
  { Declare em uses untTabela }
  Inc(numLigacoes);
  SetLength(vCanvas, numLigacoes);
  vCanvas[numLigacoes - 1] := TCanvas.Create;
  vCanvas[numLigacoes - 1].Handle := GetDC(Handle);
  vCanvas[numLigacoes - 1].Pen.Color := clBlack;
  vCanvas[numLigacoes - 1].Pen.Width := 2;
  MontaGrade;
  for i := 0 to Pred(tabelaDestino.ChavesEstrangeiras.
    Count) do
  begin
    if tabelaOrigemAdicionada(tabelaDestino.
      ChavesEstrangeiras[i], tabelaOrigem) then
    begin
      if tabelaOrigem.Left <= tabelaDestino.Left then
      begin
        vCanvas[numLigacoes - 1].MoveTo((
          tabelaOrigem.Left + tabelaOrigem.Width),
          (tabelaOrigem.Top +
          Trunc(tabelaOrigem.Height/2)));
        vCanvas[numLigacoes - 1].LineTo(
          tabelaDestino.Left, (tabelaDestino.Top +
          Trunc(tabelaDestino.Height/2)));
        vCanvas[numLigacoes - 1].Ellipse((
          tabelaDestino.Left - 6), (tabelaDestino.Top+
          (Trunc(tabelaDestino.Height/2) - 4)),
          (tabelaDestino.Left + 2), (tabelaDestino.Top+
          (Trunc(tabelaDestino.Height/2)) + 4));
      end
      else
      begin
        vCanvas[numLigacoes - 1].MoveTo(
          tabelaOrigem.Left, (tabelaOrigem.Top + Trunc(
          tabelaOrigem.Height/2)));
        vCanvas[numLigacoes - 1].LineTo((
          tabelaDestino.Left + tabelaDestino.Width),
          (tabelaDestino.Top + Trunc(
          tabelaDestino.Height/2)));
        vCanvas[numLigacoes - 1].Ellipse(((
          tabelaDestino.Left + tabelaDestino.Width) +
          6), (tabelaDestino.Top + (Trunc(
          tabelaDestino.Height/2) + 4)),
          (tabelaDestino.Left + tabelaDestino.Width) -
          2, (tabelaDestino.Top + (Trunc(
          tabelaDestino.Height/2)) - 4));
      end;
    end;
  end;
end;
```

Listagem 6. Método tabelaOrigemAdicionada

```
function TFrmRegiao.tabelaOrigemAdicionada(
  const nomeTabela: string;
  var tabelaOrigem: TTabela): Boolean;
var
  i: integer;
begin
  Result := False;
  for i := 0 to Pred(ComponentCount) do
    if (Components[i] is TTabela) and ((
      Components[i] as TTabela).Tabela =
      nomeTabela) then
    begin
      tabelaOrigem := Components[i] as TTabela;
      Result := True;
      Break;
    end;
end;
```

Volte ao formulário *FrmPrincipal* e declare as seguintes variáveis na seção *private*:

```
private
  posX, posY: Integer;
  numTabelas: Integer;
  tabelaSelecionada: TTabela;
  nomeER: string;
```

Utilizaremos as variáveis *posX* e *posY* para armazenar a posição da última tabela adicionada no diagrama, isso é o *Top* e o *Left* a ser utilizado como referência para a inclusão de uma nova tabela. A variável *numTabelas* armazenará o número total de tabelas e *views* presentes no diagrama ativo.

A variável *tabelaSelecionada* que é do tipo *TTabela* (adicione no uses a unit criada anteriormente) armazenará o objeto referente à tabela selecionada pelo usuário dentro do digrama. Por último, a variável *nomeER* terá o nome do digrama quando o mesmo for salvo pela primeira vez pelo usuário. Dê um duplo clique no *aclPrincipal* para abrir o editor de *actions* e adicione algumas ações, como mostra a **Figura 4**.

Crie um método chamado *ConfigurarBD* e implemente-o como mostra a **Listagem 7**.

Listagem 7. Código pata configurar o banco de dados

```
procedure TFrmPrincipal.ConfigurarBD;
begin
  FrmBD := TFrmBD.Create(Self);
  try
    if FrmBD.ShowModal = mrOK then
    begin
      if not conexao.Connected then
        conexao.Connected := True;
      sbInfo.Panels[2].Text :=
        'Banco de dados: ' + conexao.Params.Values[
        'Database'];
    end;
  finally
    FrmBD.Release;
    FrmBD := nil;
  end;
end;
```

Selecione *acConfigurarBD* no *aclPrincipal* e faça a chamada ao método *ConfigurarBD* em seu evento *OnExecute*. Declare a seguinte variável na seção *public* do *FrmPrincipal*:

```
public
  VisualizarRelacionamentos: Boolean;
```

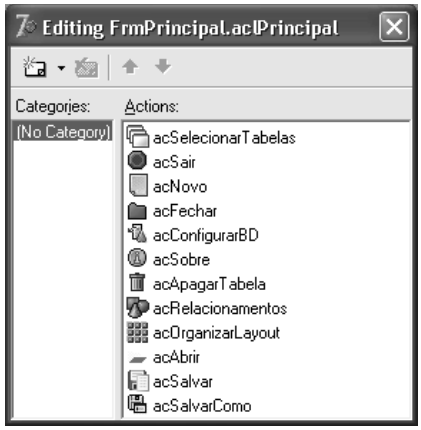


Figura 4. Actions responsáveis pelas ações dos menus e botões do formulário principal

Utilizaremos essa variável para controlar se os relacionamentos entre as tabelas devem ou não ser mostrados ao usuário. Crie um método chamado *DiagramaVazio* e declare-o também dentro da seção *public*. Veja a implementação do método na **Listagem 8**.

Crie um novo método chamado *NovoProjeto* e implemente-o com o código da **Listagem 9**.

Listagem 8. Método DiagramaVazio

```
function TFrmPrincipal.DiagramaVazio: Boolean;
var
  i: Integer;
begin
  Result := True;
  for i := 0 to Pred(FrmRegiao.ComponentCount) do
    if (FrmRegiao.Components[i] is TTabela) then
    begin
      Result := False;
      Exit;
    end;
end;
```

Listagem 9. Método NovoProjeto

```
procedure TFrmPrincipal.NovoProjeto;
var
  i, j : Integer;
begin
  posX := 15;
  posY := 15;
  numTabelas := 0;
  tabelaSelecionada := nil;
  nomeER := '';
  Self.Caption := 'ERPlus';
  VisualizarRelacionamentos := True;
  acRelacionamentos.Checked := True;
  FrmRegiao.Color := clWhite;
  for i := 0 to mmPrincipal.Items.Count - 1 do
  begin
    mmPrincipal.Items[i].Enabled := True;
    for j := 0 to mmPrincipal.Items[i].Count - 1 do
      mmPrincipal.Items[i].Items[j].Enabled := True;
  end;
  for i := 0 to tbPrincipal.ButtonCount - 1 do
    tbPrincipal.Buttons[i].Enabled := True;
  for i := 0 to pmRegiao.Items.Count - 1 do
    pmRegiao.Items[i].Enabled := True;
  ConfigurarBD;
end;
```

Ainda dentro do *Code Editor*, crie mais um método, agora de nome "FecharProjeto", e implemente-o com o código da **Listagem 10**.

Listagem 10. Método para fechar o projeto ativo

```
procedure TFrmPrincipal.FecharProjeto;
var
  i: Integer;
begin
  Screen.Cursor := crHourGlass;
  for i := FrmRegiao.ComponentCount - 1 downto 0 do
    if (FrmRegiao.Components[i] is TTabela) then
      (FrmRegiao.Components[i] as TTabela).Free;
  FrmRegiao.Repaint;
  FrmRegiao.Color := clBtnFace;
  sbInfo.Panels[2].Text := '';
  sbInfo.Panels[3].Text := '';
  sbInfo.Panels[4].Text := '';
  Self.Caption := 'ERPlus';
  FrmRegiao.Width := sbRegiao.Width - 2;
  FrmRegiao.Height := sbRegiao.Height - 2;
  mmPrincipal.Items[1].Enabled := False;
  mmPrincipal.Items[2].Enabled := False;
  mmPrincipal.Items[3].Items[0].Enabled := False;
  mmPrincipal.Items[3].Items[1].Enabled := False;
  acRelacionamentos.Checked := False;
  for i := 2 to mmPrincipal.Items[0].Count - 1 do
    mmPrincipal.Items[0].Items[i].Enabled := False;
  for i := 0 to tbPrincipal.ButtonCount - 1 do
    if (tbPrincipal.Buttons[i].Name <> 'tbNovo') and
       (tbPrincipal.Buttons[i].Name <> 'tbAbrir') and
       (tbPrincipal.Buttons[i].Name <> 'tbSobre') and
       (tbPrincipal.Buttons[i].Name <> 'tbSair') then
      tbPrincipal.Buttons[i].Enabled := False;
  for i := 0 to pmRegiao.Items.Count - 1 do
    pmRegiao.Items[i].Enabled := False;
  with conexao do
  begin
    if Connected then
      Connected := False;
    Params.Values['Database'] := '';
    Params.Values['User_Name'] := '';
    Params.Values['Password'] := '';
  end;
  Screen.Cursor := crDefault;
end;
```

Faça a chamada a esses métodos a partir das *actions* (*acNovo* e *acFechar*) definidas dentro do *aclPrincipal*. Para cada *action* implementada vincule a mesma ao seu respectivo item de menu do *mmPrincipal* e ao botão na *ToolBar* utilizando a propriedade *Action* desses componentes.

Adicione também nas *actions* as chamadas para os formulários de selecionar as tabelas (*FrmTabelas*) e configurar o banco de dados (*FrmBD*). Para o formulário de selecionar tabelas, adicione o código da **Listagem 11** na *action* correspondente (*acSelecionarTabelas*).

Adicione na *unit* mais um método com o nome de "CampoChavePrimaria" e implemente-o como mostra a **Listagem 12**.

Listagem 11. Chama o formulário para selecionar tabelas

```
var
  i, j: Integer;
begin
  if not conexao.Connected then
  begin
    MessageDlg('Configure o banco de dados!',
      mtInformation, [mbOk], 0);
    Exit;
  end;
  FrmTabelas := TFrmTabelas.Create(Self);
  try
    conexao.GetTableNames(FrmTabelas.lbTabelas.Items, False);
    for i := 0 to FrmRegiao.ComponentCount - 1 do
    begin
      if FrmRegiao.Components[i] is TTabela then
      begin
        for j := FrmTabelas.lbTabelas.Items.Count - 1
          downto 0 do
        begin
          if (FrmRegiao.Components[i] as TTabela).Tabela =
            FrmTabelas.lbTabelas.Items[j] then
            FrmTabelas.lbTabelas.Items.Delete(j);
        end;
      end;
    end;
    FrmTabelas.Caption := 'Tabelas disponíveis (' + IntToStr(
      FrmTabelas.lbTabelas.Items.Count) + ')';
    if FrmTabelas.ShowModal = mrOk then
    begin
      Screen.Cursor := crHourGlass;
      for i := 0 to Pred(
        FrmTabelas.lbTabelas.Items.Count) do
      begin
        if FrmTabelas.lbTabelas.Selected[i] then
        begin
          Application.ProcessMessages;
          AdicionarTabela(
            FrmTabelas.lbTabelas.Items[i]);
        end;
      end;
      Screen.Cursor := crDefault;
      if FrmPrincipal.DiagramaVazio then
      begin
        FrmRegiao.Canvas.FillRect(FrmRegiao.ClientRect);
        FrmRegiao.MontaGrade;
      end
      else
      begin
        { Loop para adicionar as ligações entre as
          tabelas adicionadas no diagrama }
        if VisualizarRelacionamentos then
        begin
          FrmRegiao.PrepararLigacoes;
          for i := 0 to pred(
            FrmRegiao.ComponentCount) do
          begin
            if (
              FrmRegiao.Components[i] is TTabela) then
              FrmRegiao.CriarLigacao((
                FrmRegiao.Components[i] as TTabela));
          end;
        end;
      end;
    end;
  finally
    FrmTabelas.Release;
    FrmTabelas := nil;
  end;
end;
```

Listagem 12. Verificando se o campo é chave primária da tabela

```
function TFrmPrincipal.CampoChavePrimaria(
  const tabela, campo: string): Boolean;
begin
  Result := False;
  with datasetAux do
  begin
    Close;
    CommandText := 'SELECT S.RDB$FIELD_NAME ' +
      '  FROM RDB$INDEX_SEGMENTS S, RDB$INDICES I, ' +
      '  RDB$RELATION_CONSTRAINTS R ' +
      ' WHERE S.RDB$INDEX_NAME = I.RDB$INDEX_NAME ' +
      '   AND I.RDB$RELATION_NAME = ' +
      '   R.RDB$RELATION_NAME ' +
      '   AND I.RDB$INDEX_NAME = R.RDB$INDEX_NAME ' +
      '   AND R.RDB$RELATION_NAME = ' +
      QuotedStr(tabela) +
      '   AND R.RDB$CONSTRAINT_TYPE = ' +
      QuotedStr('PRIMARY KEY');
    Open;
    First;
    while not Eof do
    begin
      if AnsiUpperCase(Trim(Fields[0].AsString)) =
        AnsiUpperCase(campo) then
      begin
        Result := True;
        Break;
      end;
      Next;
    end;
    Close;
  end;
end;
```

Esse método executará uma instrução SQL nas tabelas de sistema do InterBase ou Firebird para identificar o(s) campo(s) que são chave primária da tabela informada.

Crie agora um método semelhante chamado "CampoChaveEstrangeira". Esse por sua vez será responsável por identificar através de SQL nas tabelas de sistema, os campos que possuem relacionamentos. Veja sua implementação na **Listagem 13**.

Listagem 13. Verificando as chaves estrangeiras da tabela

```
function TFrmPrincipal.CampoChaveEstrangeira(
  const tabela, campo: string;
  var foreignKey: string): Boolean;
begin
  Result := False;
  with datasetAux do
  begin
    Close;
    CommandText := 'SELECT S.RDB$FIELD_NAME, ' +
      '  R.RDB$RELATION_NAME, I.RDB$FOREIGN_KEY ' +
      '  FROM RDB$INDEX_SEGMENTS S, RDB$INDICES I, ' +
      '  RDB$RELATION_CONSTRAINTS R ' +
      ' WHERE S.RDB$INDEX_NAME = I.RDB$INDEX_NAME ' +
      '    AND I.RDB$RELATION_NAME = ' +
      '    R.RDB$RELATION_NAME ' +
      '    AND I.RDB$INDEX_NAME = R.RDB$INDEX_NAME ' +
      '    AND R.RDB$RELATION_NAME = ' + QuotedStr(tabela) +
      '    AND R.RDB$CONSTRAINT_TYPE = ' + QuotedStr('FOREIGN KEY');
    Open;
    First;
    while not Eof do
    begin
      if AnsiUpperCase(Trim(Fields[0].AsString)) =
      AnsiUpperCase(campo) then
      begin
        { método que retorna a tabela }
        { Implementado no código-fonte para download }
        foreignKey := retornaTabelaPrimaria(
          AnsiUpperCase(Trim(Fields[2].AsString)));
        Result := True;
        Break;
      end;
      Next;
    end;
    Close;
  end;
end;
```

Vamos agora implementar um dos principais métodos da aplicação, a *AdicionarTabela*, que será responsável em adicionar as tabelas do banco de dados dentro do diagrama. Veja sua implementação na **Listagem 14**.

Esse método utiliza a classe *TTabela* para carregar os campos e seus respectivos tipos dentro do *FrmRegiao* que será aberto dentro do *ScrollBox* do *FrmPrincipal*. Repare que o método *AdicionarTabela* utiliza o método auxiliar chamado *RetornaTipoCampo* que é responsável por fazer a busca do tipo do campo informado no parâmetro de entrada dentro das tabelas de sistema do IB/FB.

Nesse ponto da aplicação já podemos criar um diagrama, configurar o banco de dados e adicionar as tabelas e seus relacionamentos dentro do digrama. Para que o usuário possa apagar uma tabela do diagrama vamos implementar o método *ApagarTabelaSelecionada*, que receberá como parâmetro de entrada a tecla pressionada pelo usuário dentro da aplicação. Veja sua implementação na **Listagem 15**.

Faça a chamada a esse método no evento *OnKeyDown* do *FrmPrincipal* e dentro do evento *OnExecute* do *acApagarTabela*. Adicione ainda ao evento *OnCreate* do *FrmPrincipal* o código da **Listagem 16**.

Listagem 16. OnCreate do FrmPrincipal

```
procedure TFrmPrincipal.FormCreate(Sender: TObject);
begin
  FrmRegiao := TFrmRegiao.Create(Self);
  FrmRegiao.Top := 0;
  FrmRegiao.Left := 0;
  FrmRegiao.Width := sbRegiao.Width - 2;
  FrmRegiao.Height := sbRegiao.Height - 2;
  FrmRegiao.Parent := sbRegiao;
  FrmRegiao.Show;
  { GetVersaoArq é uma função que retorna a versão
    do projeto, e esta implementada no código-fonte para download }
  sbInfo.Panels[1].Text := 'Versão ' + GetVersaoArq;
  FecharProjeto;
end;
```

Listagem 14. Método para adicionar a tabela no diagrama

```
procedure TFrmPrincipal.AdicionarTabela(const tabela: string);
var
  vTabela: TTabela;
  campos: TStringList;
  chaveEstrangeira: string;
  i: Integer;
begin
  { Declare no uses FrmRegiao }
  vTabela := TTabela.Create(FrmRegiao);
  vTabela.Campos.StateImages := imglTabela;
  campos := TStringList.Create; inc(numTabelas);
  vTabela.Parent := FrmRegiao;
  vTabela.Tabela := tabela;
  conexao.GetFieldNames(tabela, campos);
  for i := 0 to Pred(campos.Count) do
  begin
    with vTabela.Campos.Items.Add do
    begin
      if CampoChavePrimaria(tabela, campos[i]) then
        StateIndex := 0;
      if CampoChaveEstrangeira(tabela, campos[i], chaveEstrangeira) then
      begin
        if (chaveEstrangeira <> '') then
        begin
         if vTabela.ChavesEstrangeiras.IndexOf(chaveEstrangeira) = -1 then
            vTabela.ChavesEstrangeiras.Add(
              chaveEstrangeira);
        end;
        if StateIndex = 0 then
          StateIndex := 2
        else
          StateIndex := 1;
      end;
      Caption := campos[i];
      SubItems.Add(RetornaTipoCampo(tabela, campos[i]));
    end;
  end;
  if ((numTabelas - 1) > 0) and (((numTabelas - 1)
    mod 4) = 0) then
  begin
    posX := posX + 240;
    posY := 15;
  end;
  vTabela.Top := posX;
  vTabela.Left := posY;
  posY := posY + 280;
end;
```

Listagem 15. Método para apagar uma tabela do diagrama

```
procedure TFrmPrincipal.ApagarTabelaSelecionada(const Key: Word);
var
  i: Integer;
begin
  if (Key = 46) then
    if Assigned(tabelaSelecionada) then
    begin
      if MessageDlg('Apagar tabela "' +
        tabelaSelecionada.Tabela + '" do diagrama?',
        mtConfirmation, [mbYes, mbNo], 0) = mrYes then
      begin
        for i := Pred(FrmRegiao.ComponentCount)
          downto 0 do
        begin
          if (FrmRegiao.Components[i] is TTabela) and
             ((FrmRegiao.Components[i] as TTabela).
             Tabela = tabelaSelecionada.Tabela) then
          begin
            (FrmRegiao.Components[i] as TTabela).Free;
            tabelaSelecionada := nil;
            sbInfo.Panels[3].Text := '';
            Break;
          end;
        end;
        if not VisualizarRelacionamentos then
        begin
          FrmRegiao.MontaGrade;
          Exit;
        end;
        FrmRegiao.Repaint;
        FrmRegiao.PrepararLigacoes;
        for i := 0 to pred(FrmRegiao.ComponentCount) do
        begin
          if (FrmRegiao.Components[i] is TTabela) then
            FrmRegiao.CriarLigacao((
            FrmRegiao.Components[i] as TTabela));
        end;
      end;
    end
    else
      MessageDlg('Não há tabela selecionada no '+
        'diagrama!', mtInformation, [mbOK], 0);
end;
```

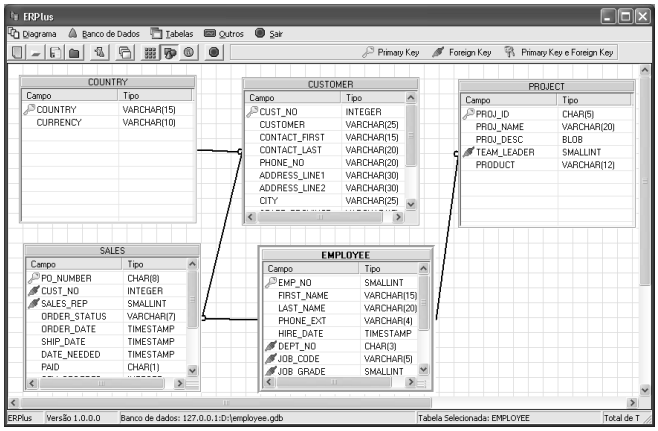


Figura 5. Visualizando as tabelas e relacionamentos do banco de dados Employee.gdb

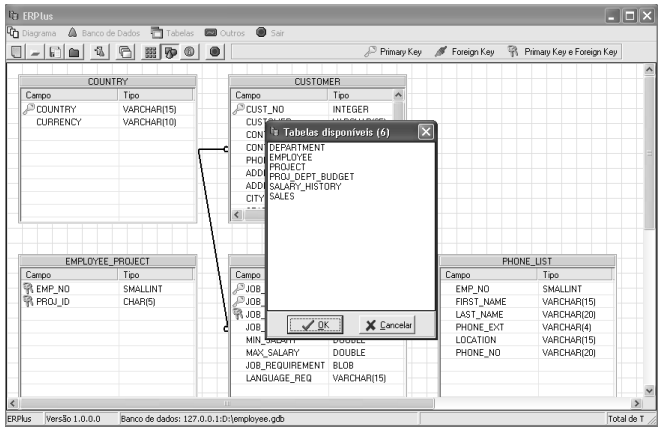


Figura 6. Selecionando as tabelas a serem incluídas no diagrama ER

No código da listagem anterior criamos o *FrmRegiao* e atribuímos como seu *Parent* o *ScrollBox* (*sbRegiao*) do *FrmPrincipal*. Para salvar o diagrama criado pelo usuário o aplicativo gera um arquivo XML contendo todas as informações necessárias para que o diagrama possa ser aberto posteriormente.

Para salvar e carregar o arquivo XML utilizamos os *ClientDataSets*: *cdsERConfig* e *cdsERTabelas* adicionados no *FrmPrincipal*.

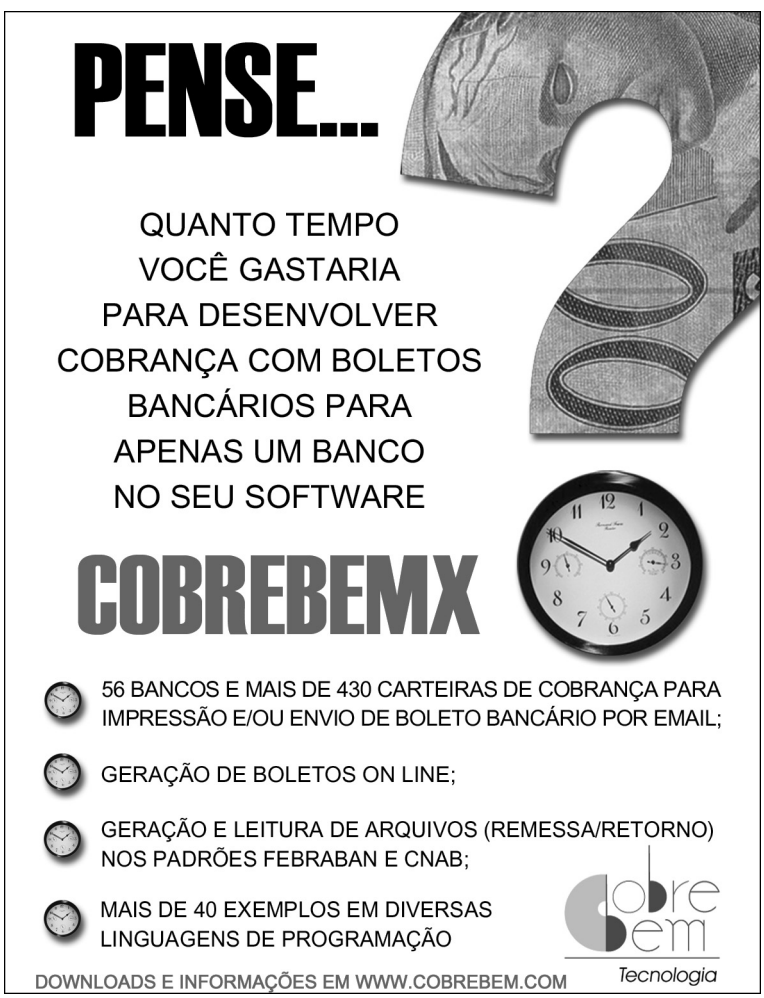
Nota: Como comentado no início, alguns métodos foram ocultados por questões de espaço. Você pode verificar todos os métodos necessários para o funcionamento do *ERPlus* no código-fonte disponível no site da ClubeDelphi.

Veja o aplicativo em execução nas **Figuras 5** e **6**.

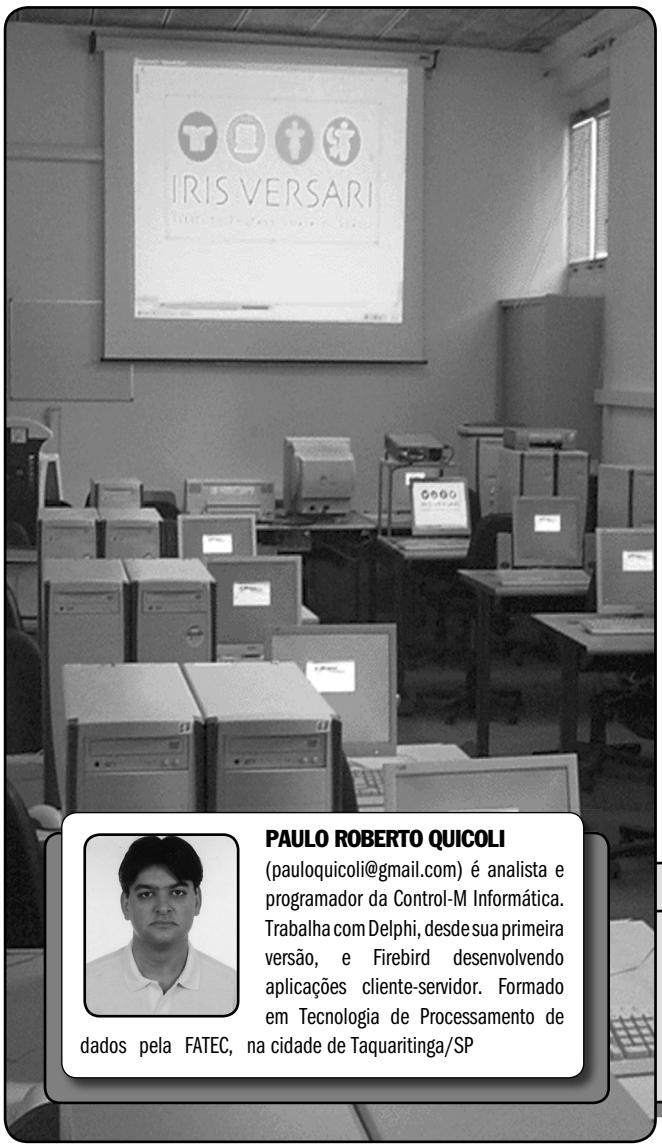
Nota: Caso você esteja utilizando o Firebird, e ao selecionar um banco de dados local receba a mensagem de erro *unavailable database*, basta adicionar o IP local (127.0.0.1) ou "localhost" na frente do caminho do banco de dados. Por exemplo: "127.0.0.1:C:\Employee.FDB".

Conclusão

Vimos neste artigo como criar uma aplicação para realizar operações de engenharia-reversa em bancos de dados InterBase e Firebird. Essa é apenas a primeira versão do *ERPlus*; você pode facilmente, implementar novas funcionalidades em cima do código existente, como opções para criação de novas tabelas no banco de dados, visualização de propriedades de cada objeto ou ainda geração de scripts DDL. Um grande abraço e até o próximo artigo. ■



Mini-Curso de Delphi e UML – Parte II



PAULO ROBERTO QUICOLI
(pauloquicoli@gmail.com) é analista e programador da Control-M Informática. Trabalha com Delphi, desde sua primeira versão, e Firebird desenvolvendo aplicações cliente-servidor. Formado em Tecnologia de Processamento de dados pela FATEC, na cidade de Taquaritinga/SP

Continuando nosso mini-curso, veremos como utilizar um framework de persistência de objetos, chamado *InstantObjects*. Também aprenderemos a criar diagramas de classes no ModelMaker, adicionar atributos e criar um relacionamento de herança entre classes.

InstantObjects

Para fazer persistência de objetos, vamos utilizar o *InstantObjects*. Acesse *www.instantobjects.org* e faça o download (arquivo *InstantObjects-2.0.rc1-src.zip* no título *Download*). Após baixar e descompactar o arquivo, abra o grupo de projeto *RunTimePackages.bpg* que está em *<instalação>\Source\PackageGroups\D7*. Caso você não esteja vendo o *Project Manager* do Delphi 7 vá no menu *View>Project Manager* para exibi-lo (**Figura 1**).

Devemos agora efetuar um *build* nos projetos, usando o menu *Project>Build All Projects*, porém antes devemos nos certificar através de um clique duplo, que o projeto em foco deve ser o *IOCore_D7.bpl*, pois os demais pacotes dependem dele.

Localizado na mesma pasta existe o grupo *DesignTimePackages.bpg* (**Figura 2**), onde vamos repetir os passos descritos anteriormente.

CLUBEDELPHI PLUS!

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como criar diagramas da classes no Delphi 2006.

http://www.devmedia.com.br/articles/visualizacomponente2.asp?comp=2369

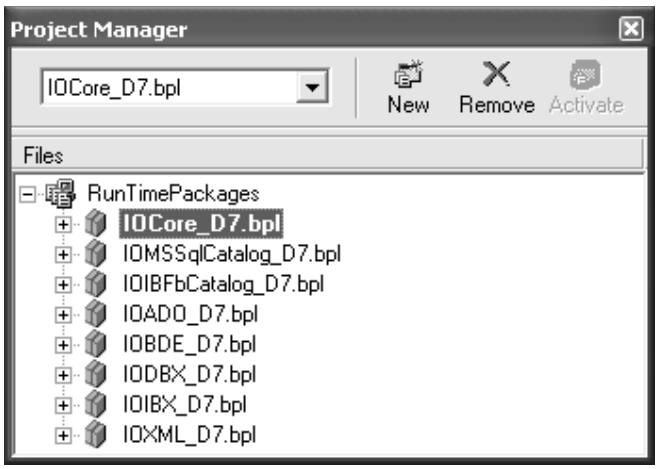


Figura 1. Pacotes de runtime do IO

Como os pacotes são *design time* é preciso que sejam instalados no IDE, para isso vamos clicar com o botão direito do mouse em cada um, a partir do primeiro, e escolher a opção *Install*. Com a instalação dos pacotes no IDE, temos um novo grupo de componentes, o *InstantObjects* (**Figura 3**).

E para terminar, devemos informar através do menu *Tools>Environment Options>Library>Library Path*, qual é a localização dos arquivos do IO. Como instalamos todos os pacotes, temos que adicionar todos os diretórios:

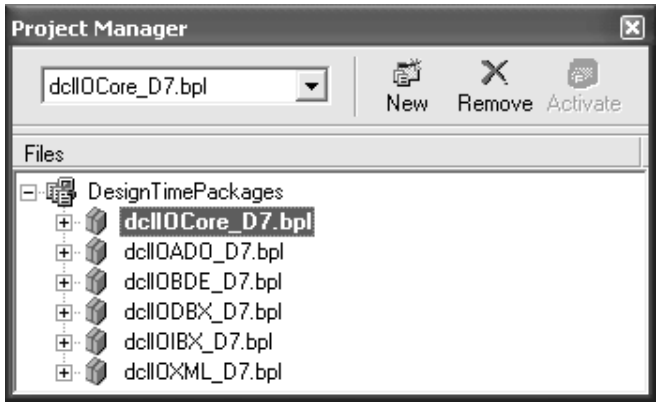


Figura 2. Pacotes de design time do IO

Figura 3. Guia de componentes do InstantObjects 2

```
<Instalação>/Source/Core
<Instalação>/Source/Brokers/<nome do broker>
<Instalação>/Source/Catalogs/<nome do catalogo>
```

Veja na **Listagem 1** a lista completa das pastas que devem ser adicionadas.

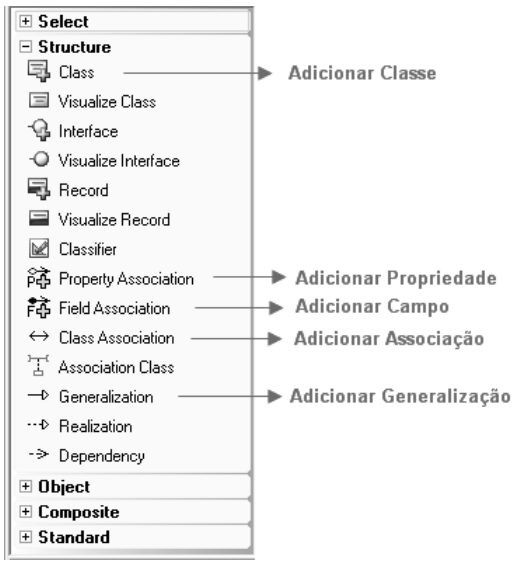


Figura 4. Itens de um diagrama de classe

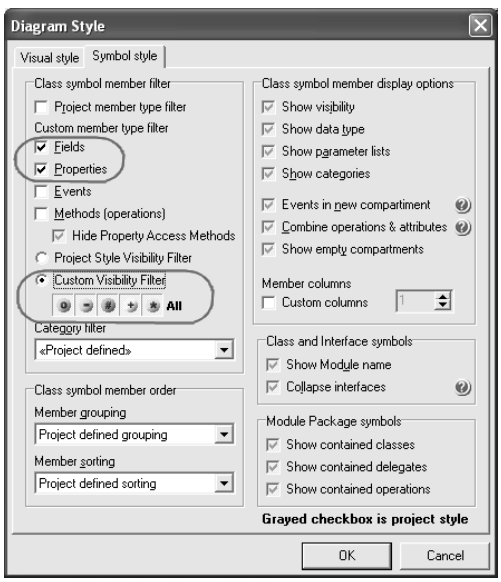


Figura 5. Configuração de exibição das classes

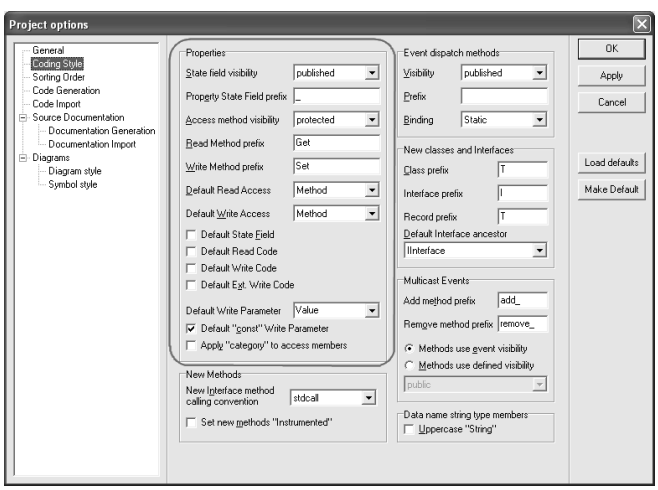


Figura 6. Configurando a geração de código

Listagem 1. Diretórios a serem adicionados no Path Library do Delphi

```
<Instalação>/Source/Core
<Instalação>/Source/Brokers/ADO
<Instalação>/Source/Brokers/BDE
<Instalação>/Source/Brokers/DBX
<Instalação>/Source/Brokers/XML
<Instalação>/Source/Catalogs/IBFb
<Instalação>/Source/Catalogs/MySql
```

Diagrama de classes

Abra o arquivo do ModelMaker que contém os diagramas de Caso de Uso construídos na edição anterior. Clique no botão novo diagrama e escolha *Add Class Diagram*, e o renomeie para "Classes de Negócio".

Veja que a barra de objetos de diagrama exibe agora os elementos que compõem um diagrama de classes. Na **Figura 4**, temos as descrições dos símbolos mais simples, pois uma explanação maior sobre diagramas de classe foge do nosso contexto.

Antes de começar, vamos configurar o ModelMaker para que possa gerar as classes da forma que o *InstantObjects* precisa e também para que toda a classe seja exibida. O *InstantObjects* requer que as propriedades de nossas classes sejam publicadas (*published*) e sejam acessadas por métodos *Get/Set*.

Por sua vez, esses métodos devem ler/gravar valores em campos também publicados, porém com um detalhe, o nome desses campos deve ser o sinal "_" (*underline*), mais o nome da propriedade a que se refere. Por exemplo, se temos uma propriedade chamada "Nome", o campo referente a ela deve ser "_Nome".

Clique na área do diagrama com o botão direito do mouse, escolha a opção *Diagram>Diagram Properties* e configure a guia *Symbol Style* conforme a **Figura 5**.

Agora, acesse o menu *File>Project Properties*, guia *Coding Style*. Essa guia oferece configurações para que possamos ajustar a forma como será gerado o código pelo ModelMaker. Configure conforme a **Figura 6**.

É necessário também importar a classe *TInstantObject*, que servirá de classe base para nossas classes, ela está contida na unit *IntantPersistence.pas*. Clique na guia *Modules*, com o botão direito do mouse na área vazia, escolha o item *Import Source File*. Será mostrada uma janela, escolha *VCL 7 Source*.

Localize a unit *InstantPersistence.pas*, que por padrão é encontrada em *<Instalação>\Source\Core* (**Figura 7**).

Clique no botão *Interfaces* e confirme, clicando em *Abrir*. Após, é necessário escolher qual classe que vamos adicionar ao nosso diagrama, nesse exemplo a classe *TInstantObject* (**Figura 8**). A partir de agora a classe *TInstantObject* está disponível para que seja utilizada em nosso diagrama.

Criando o digrama de classes

Clique na guia *Classes* e clique com o botão direito, escolhendo a opção *Add Class*. Na janela de definição da classe, informe em *Class Name* "TRevista" e em *Ancestor* escolha *TInstantObject*. Com a classe inserida no modelo, podemos adicionar campos, propriedades, métodos. O ModelMaker oferece de forma fácil acesso a essas funcionalidades. Veja na **Figura 9** uma breve descrição.

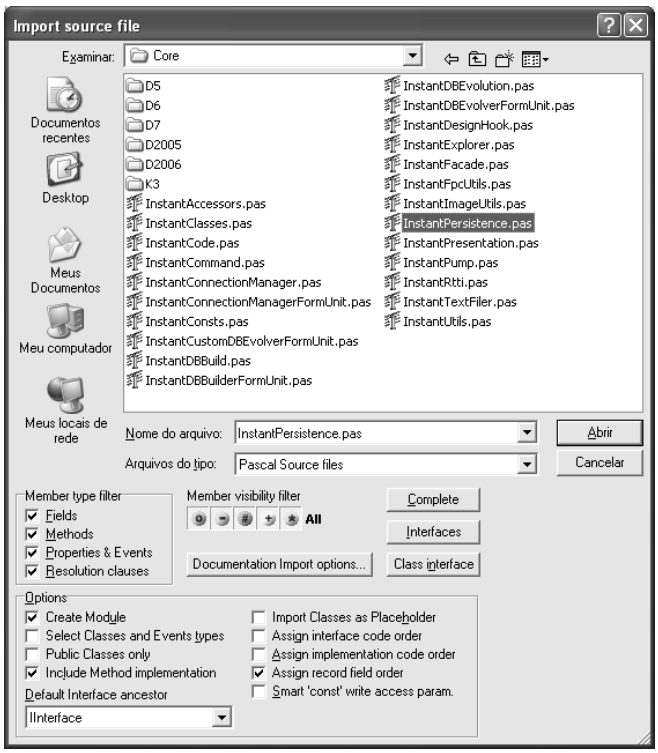


Figura 7. Importando unit InstantPersistence.pas

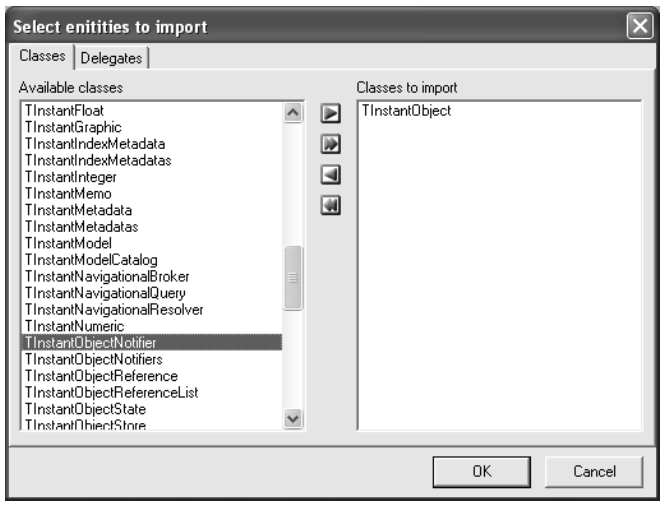


Figura 8. Importando a classe TInstantObject

Clique no botão [P+] e vamos adicionar uma nova propriedade. Em *Name* digite "Nome" e em *Type Presets* escolha *string* e clique em OK. Como já explicado, cada propriedade terá um campo publicado correspondente. Clique no botão [F+] para adicionar um novo campo e configure-o conforme a **Figura 10**.

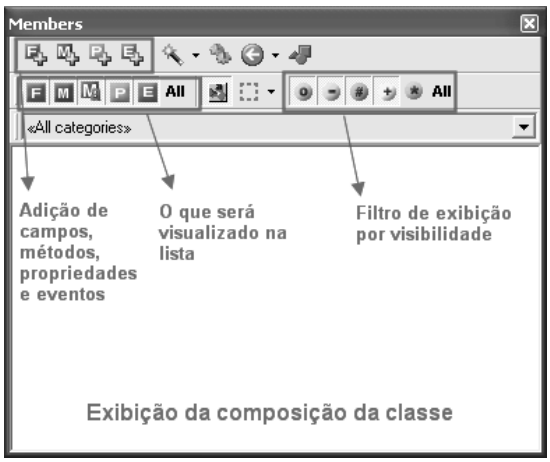


Figura 9. Janela de manutenção de classes

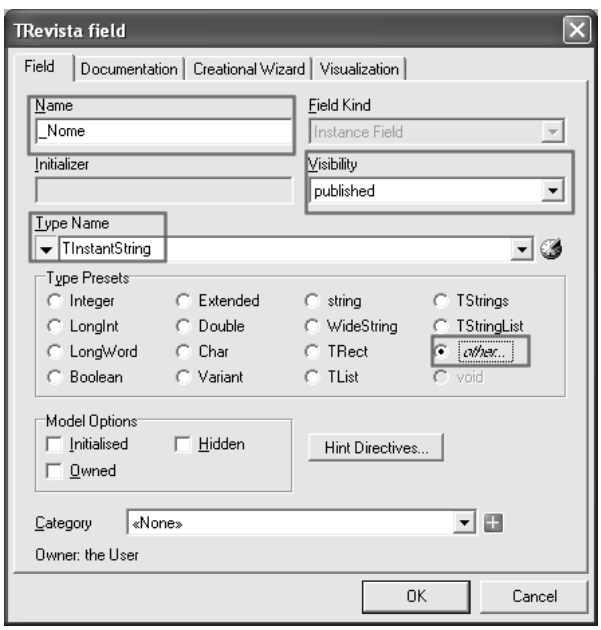


Figura 10. Inserindo um campo

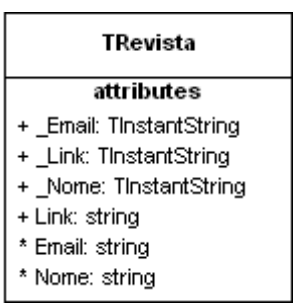


Figura 11. Classe TRevista no Diagrama

Propriedade	Tipo	Campo	Tipo
Nome	String	_Nome	TInstantString
NumeroEdicao	Integer	_NumeroEdicao	TInstantInteger
Resumo	String	_Resumo	TInstantMemo
PalavrasChave	String	_PalavrasChave	TInstantString

Tabela 1. Estrutura da classe TArtigo

Faça o mesmo para as propriedades: "Email" e "Link". Na **Figura 11**, vemos como nossa classe ficou (abra o Diagrama *Classes de Negócio* e arraste a classe para dentro).

Crie agora a classe "TArtigo" adicionando as propriedades constantes na **Tabela 1**.

Agora é preciso fazer a ligação entre a classe *TArtigo* e *TRevista*, conforme o Caso de Uso número 1 (visto na edição anterior). Clique sobre a seta *Property Association* e arraste de *TArtigo* para *TRevista* e preencha as opções conforme a **Figura 12**.

E adicione seu respectivo campo "_Revista" com o tipo *TInstantReference*, na classe *TArtigo*. Com isso, temos um diagrama de classes conforme visto na **Figura 13**.

Conclusão

Após uma introdução ao ModelMaker vimos como é fácil desenvolver uma documentação UML sem morosidade. Vimos como os Casos de Uso são importantes, pois serviram de base para a construção e orientação no nosso Digrama de Classes. No próximo artigo veremos como gerar código Delphi a partir desse diagrama, para começar com a programação do nosso exemplo. Até lá. ■

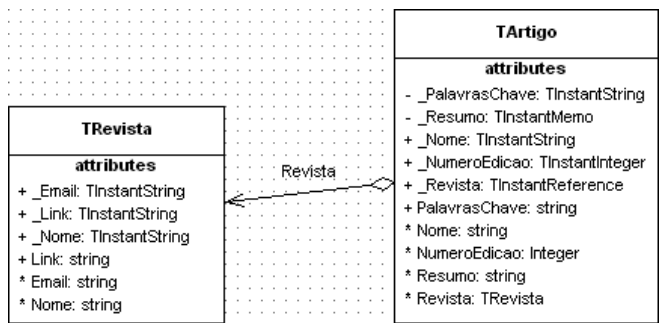


Figura 13. Resultado Final

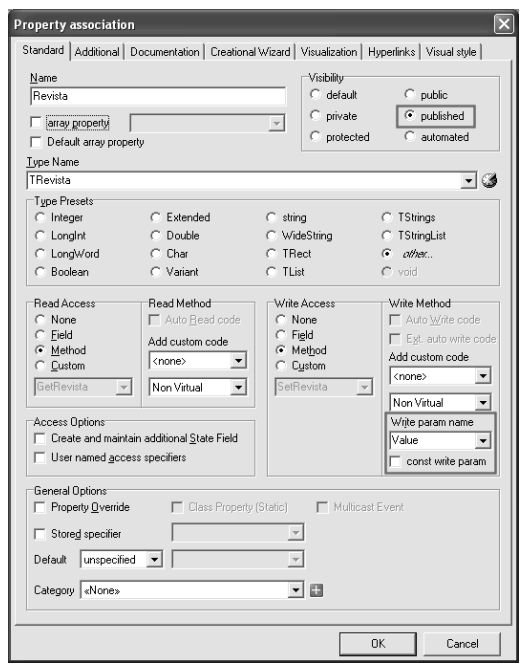
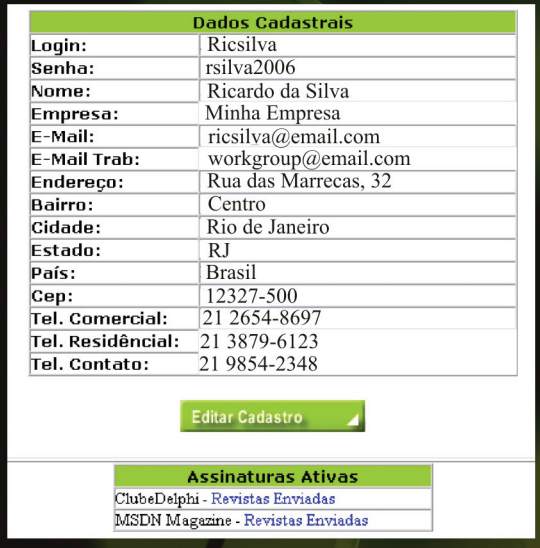


Figura 12. Associando TRevista à TArtigo

Fale conosco!

Conheça a Central de Atendimento on-line da DevMedia. Agora ficou muito mais fácil de acompanhar sua assinatura, alterar seus dados cadastrais e retirar qualquer dúvida sobre nossos serviços e produtos. Para isso basta ter em mãos seu login e senha.



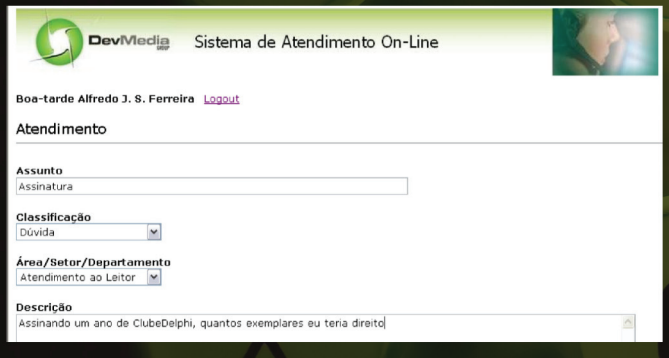
Na Central de Atendimento você:

- Consulta a data de envio de cada exemplar

- Consulta as perguntas mais frequentes

- Renova por boleto ou cartão de crédito

- Consulta e altera seus dados cadastrais



Para entrar em contato com a DevMedia, basta postar sua dúvida na própria Central de Atendimento através de um sistema web, prático, seguro e fácil de usar. Com essa nova ferramenta você tem a garantia de receber suas respostas em menor tempo e com alto padrão de qualidade.

www.devmedia.com.br/central

DevMedia GROUP

Quantas cópias de seu software existem no mercado?

Esta é a chave mais segura do mundo contra pirataria de software.

Comprove você mesmo!

SafeNet
The Foundation of Information Security

Alameda Tocantins, 280 - Barueri-SP

Tel: +55 11 4208-7700

http://br.safenet-inc.com - Contato: infobrasil@safenet-inc.com